



**Python -
это
интересно!**

Содержание

Введение.....	2
Программирование в Excel (openpyxl - библиотека связи Python и Excel).....	3
Создание игр на Python (библиотека PyGame).....	14
Пример программы «Змейка».....	15
Пример программы «Tetris».....	17
Создание сайтов: фреймворк Django.....	27

Введение

Согласно индексу [TIOBE](#), на декабрь 2025 года наиболее упоминаемыми языками стали Python, C и C++.

Dec 2025	Dec 2024	Change	Programming Language	Ratings	Change
1	1		 Python	23.64%	-0.21%
2	4	▲	 C	10.11%	+1.01%
3	2	▼	 C++	8.95%	-1.87%
4	3	▼	 Java	8.70%	-1.02%
5	5		 C#	7.26%	+2.39%
6	6		 JavaScript	2.96%	-1.66%
7	9	▲	 Visual Basic	2.81%	+0.85%
8	8		 SQL	2.10%	+0.11%
9	26	▲	 Perl	1.97%	+1.33%
10	16	▲	 R	1.96%	+0.91%
11	11		 Delphi/Object Pascal	1.91%	+0.48%
12	10	▼	 Fortran	1.60%	-0.18%
13	15	▲	 MATLAB	1.52%	+0.43%
14	24	▲	 Ada	1.49%	+0.77%
15	7	▼	 Go	1.37%	-0.80%
16	12	▼	 PHP	1.36%	-0.03%
17	14	▼	 Rust	1.30%	+0.01%
18	13	▼	 Scratch	1.11%	-0.23%
19	17	▼	 Assembly language	1.04%	-0.01%
20	23	▲	 Kotlin	0.92%	+0.10%

Автор языка программирования Python - голландский программист Гвидо ван Россум. В 1982 году Гвидо ван Россум окончил университет и попал в команду разработчиков института CWI, где до 1986 года занимался проектированием языка ABC - прототипа



Python. Язык медленно распространялся и не получал оперативной обратной связи от пользователей, так как в то время не было ещё интернета. В 1987 году проект ABC закрылся. После неудачного опыта с ABC Гвидо ван Россум оценил силу сообщества. Он понял, что язык программирования не может развиваться без обратной связи, - нужна система, в которой каждый разработчик будет предлагать улучшения или голосовать за предложения других участников сообщества.

В 1986 году Гвидо оставался сотрудником CWI, но перешёл из ABC в проект Амоеба — там он разрабатывал операционную систему для крупных организаций. Амоеба объединяла сеть пользовательских компьютеров в единую станцию и оптимизировала их совместную работу над производительными задачами.

В 1989 году системе Амоеба не хватало языка сценариев, поэтому Гвидо ван Россум планировал мини-проект: собирался написать язык программирования на основе наработок ABC. В период рождественских праздников Гвидо сделал первые наброски и вскоре показал коллегам прототип будущего Python. Прототип Python понравился разработчикам CWI, и многие сразу включились в процесс: начали использовать язык для внутренних проектов и помогли доработать код.

20 февраля 1991 года Гвидо ван Россум создал дистрибутив и опубликовал код языка Python через сеть Usenet. Так появилась версия 0.9.0. Гвидо ван Россум не мог опубликовать дистрибутив без названия. Гвидо считал нейминг пустой тратой времени и поэтому назвал свой язык в честь комедийного сериала «Летающий цирк Монти Пайтона» - это одно из его любимых шоу и первая ассоциация, которая пришла ему в голову.

Язык подхватили и он стал активно развиваться и сегодня он наиболее популярный язык программирования.

Огромное количество встроенных и сторонних библиотек позволяют тем, кто знает Python, найти себя в любой области программирования.

Если Вы даже не знаете Python, Вы увидите результаты действия этих программ, приведённых здесь и сможете определиться в какой области Вам будет интересно использовать язык программирования Python.

Программирование в Excel (openpyxl - библиотека связи Python и Excel)

Если у Вас не установлена программа работы с электронными таблицами Excel Microsoft Office то, не заморачиваясь с лицензиями, можете установить LibreOffice. Выходите на сайт LibreOffice (<https://libre-office.su/download/>) и скачиваете нужную версию для Windows. Загрузится файл libreoffice_7.6.2_win_x86.msi (или другой, актуальный на момент скачивания). Щёлкните мышкой по нему чтобы начать процесс установки. На linux, как правило, этот офисный пакет уже установлен.

Теперь, что касается установки интерпретатора Python и его библиотек. На данный момент разработка новых проектов с нуля чаще всего ведётся на Python 3. Поэтому его и надо установить на ОС Windows. На linux он уже, как правило, установлен.

Действия по установке Python на ОС Windows.

Нужно перейти в меню Пуск, ввести "Microsoft Store" и щелкнуть на ссылку, чтобы открыть магазин.

Когда магазин откроется, выбирается Поиск в верхнем правом меню и нужно ввести "Python". Далее выбирается нужная версия Python». Рекомендуется использовать самую последнюю версию. Важно: установка Python из Microsoft Store содержит стандартный диспетчер пакетов pip. Pip позволяет устанавливать дополнительные пакеты, которые не входят в стандартную библиотеку Python, и управлять ими.

А, вот для linux pip может быть не установлен.

Убеждаемся, что программа pip у нас установлена:

`pip -version`

Если программа pip отсутствует, то её можно установить через терминал. Для Ubuntu или других вариантах операционных систем, основанных на пакетах deb (В Российском реестре операционных систем, таким аналогом может быть AstraLinux), это делается следующей командой:

`sudo apt-get install python3-pip`

Для любых клонов Red Hat, основанных на пакетах rpm (В Российском реестре программ - Ред ОС):

`sudo dnf install python3 python3-wheel`

Если по какой-то причине sudo не отрабатывает, тогда можно просто перейти в режим администратора, набрав в терминале:

`su`

с подтверждением пароля root, и произвести установку:

`dnf install python3 python3-wheel`

А затем выйти из root, набрав команду:

`exit`

Ещё надо установить редакторы в которых мы будем писать коды программ.

Лучше использовать кроссплатформенные редакторы, которые подходят для разных операционных систем. Можно поставить простой редактор gedit. На Windows он не стоит, но можно скачать установочный файл с сайта

<https://gedit.ru.uptodown.com/windows/download> . На Linux он уже установлен

либо с таким же названием gedit, либо с названием Pluma, в зависимости от используемого варианта операционной системы linux. Существует и более

мощный кроссплатформенный редактор для написания python-кода. Точнее, это не редактор, а целая интегрированная среда, под названием PyCharm. Загрузку и установку можно посмотреть на одноимённом сайте.

Теперь мы можем устанавливать дополнительные пакеты Python с помощью программы pip.

Но прежде, нужно сказать, что хорошей идеей будет установка дополнительных библиотек в обособленные виртуальные области или среды. Конечно, можно установить приложение только один раз, и использовать его в различных проектах. Но есть одна сложность, заключающаяся в конфликте версий. Допустим, вы установили все необходимые библиотеки Python, для какого-либо проекта. Всё работает. Затем Вам понадобилось разработать другой проект на основе обновлённых библиотек. Последний проект работает, а первый, с новыми версиями библиотек, может уже быть неисправным. Здесь спасают виртуальные среды. К каждой такой среде установлены свои версии библиотек. Всё работает независимо. Ещё большой плюс использования виртуальных сред — простое удаление того или иного проекта со всеми библиотеками. Для этого нужно только дезактивировать виртуальную среду и удалить каталог, с которого она (виртуальная среда) запускалась. Такое простое решение позволяет устанавливать различные библиотеки Python в различных виртуальных средах. И, если Вам не понравилось работать с установленной библиотекой, или область её применения Вас не заинтересовала, можно спокойно и быстро избавиться от этой библиотеки. Таким образом можно перебрать множество библиотек Python, и, поработав с очередной библиотекой, её благополучно удалить, чтобы не засорять компьютер.

Для создания виртуального окружения нужен `virtualenv`. Есть более свежий инструмент `venv`. Устанавливать `venv` не нужно - он входит в стандартную библиотеку Python.

Для создания виртуального окружения, необходимо создать нужную рабочую директорию, перейти в неё и выполнить команду:

```
python -m venv venv
```

Активация для Windows:

```
venv\Scripts\activate.bat
```

Получится типа:

```
(venv) root@purplegate:/var/test
```

Префикс `(venv)` говорит, что мы находимся в виртуальной среде.

Установка библиотеки `openpyxl` - работа Python с Excel.

Существует библиотека, объединяющая работу Python с разными форматами данных, а также их математическую обработку. Это библиотека `pandas`.

Запускаем `python` из командной строки для Windows:

```
python
```

или для linux:

```
python3
```

появится приглашение

```
>>>
```

Проверяем наличие установки `pandas`:

```
import pandas
```

Если после этого вновь появится приглашение `>>>`, значит всё нормально, `pandas` был установлен. Иначе устанавливаем `pandas` в командной строке

терминала, предварительно покинув интерпретатор Python, нажав сочетание клавиш **Ctrl+Z**:

```
pip install pandas [all]
```

Установка pandas с постфиксом [all] позволяет установить все входящие в него модули, в том числе и библиотеку openpyxl.

Ещё раз запускаем интерпретатор Python:

```
python(3)
```

и проверяем наличие библиотеки pandas:

```
import pandas
```

Теперь всё нормально, и нам снова выводится приглашение >>>.

Не выходя из интерпретатора Python, проверяем наличие библиотеки openpyxl, которая должна установиться, когда мы устанавливали pandas, используя постфикс [all].

Пишем команду по импорту библиотеки openpyxl .

```
import openpyxl
```

Ошибок нет - снова выводится приглашение >>>

Если произошла ошибка и openpyxl не установлена, то установить её можно из командной строки терминала, выйдя из Python сочетанием клавиш **Ctrl+Z**:

```
pip install openpyxl
```

И снова войдя в интерпретатор Python, набрать команду импорта openpyxl:

```
import openpyxl
```

Теперь должно быть всё в порядке.

Проверим работу openpyxl в действии.

Допустим, нужно выяснить, как поддерживается та или иная партия в различных регионах России. Но политическая партия может зависеть от региона проживания.

Сделать выборку из всей генеральной совокупности нам поможет двухступенчатый выбор - это метод взятия выборки в два этапа. Применим метод расслоенного выбора, приняв за слой район проживания. Анализировать будем на выборке из генеральной совокупности 1800 респондентов.

Данные из открытых источников нам показывают, что на 1 января 2023 года по оценке Росстата в России постоянно проживало **146 447 424** человек.

Двухступенчатый выбор

Дана таблица [python_excel.xlsx](#) (Тексты программ и данных располагаются в zip файле по адресу: pageforweb.ru/python_is_interesting.zip)

Шаг 1.

Пронумеруем наш список по диапазонам численности каждого региона.

Регион	Численность населения	Первый номер в общем списке	Последний номер в общем списке
Белгородская	1514527		

область	
Брянская область	1152505
Владимирская область	1325510
Воронежская область	2285282
Ивановская область	914725
Калужская область	1070853
Костромская область	571900
Курская область	1067034
Липецкая область	1126263
Московская область	8591736
Орловская область	700276
Рязанская область	1088918
Смоленская область	873041
Тамбовская область	966250
Тверская область	1211183
Тульская область	1481471
Ярославская область	1194605
г. Москва	13104177
Республика Карелия	527880
Республика Коми	726434
Ненецкий АО	41383
Архангельская обл. без НАО	964304
Вологодская область	1128782
Калининградская обл.	1032343
Ленинградская	2023767

область	
Мурманская	
область	658698
Новгородская	
область	575926
Псковская	
область	587786
г.Санкт-Петербург	5600044
Республика	
Адыгея	497985
Республика	
Калмыкия	264483
Республика Крым	1916805
Краснодарский	
край	5819345
Астраханская	
область	950557
Волгоградская	
область	2470057
Ростовская	
область	4164547
г. Севастополь	558273
Республика	
Дагестан	3209781
Республика	
Ингушетия	519078
Кабардино-	
Балкарская Респ.	903266
Карачаево-	
Черкесская Респ.	468444
Респ. Северная	
Осетия-Алания	680748
Чеченская	
Республика	1533209
Ставропольский	
край	2891204
Респ.	
Башкортостан	4077600
Республика	
Марий Эл	672321
Республика	771373

Мордовия	
Республика	
Татарстан	4001625
Удмуртская	
Республика	1442251
Чувашская	
Республика	1173177
Пермский край	2508352
Кировская	
область	1138112
Нижегородская	
обл.	3081817
Оренбургская	
область	1841377
Пензенская	
область	1246609
Самарская	
область	3142683
Саратовская	
область	2404944
Ульяновская	
область	1181006
Курганская	
область	761586
Свердловская	
область	4239161
Ханты-	
Мансийский АО-	
Югра	1730353
Ямало-Ненецкий	
АО	512387
Тюменская обл.	
без а.о.	1608494
Челябинская	
область	3407145
Республика Алтай	210769
Республика Тыва	337271
Республика	
Хакасия	530233
Алтайский край	2130950
Красноярский	
край	2845545

Иркутская область	2344360
Кемеровская область	2568238
Новосибирская обл.	2794266
Омская область	1832064
Томская область	1052106
Республика Бурятия	974628
Республика Саха (Якутия)	997565
Забайкальский край	992429
Камчатский край	288730
Приморский край	1820076
Хабаровский край	1284090
Амурская область	756198
Магаданская область	134315
Сахалинская область	460535
Еврейская авт. область	147458
Чукотский АО	47840

Посмотрим, как можно с помощью Excel получить Первый номер в общем списке и Последний номер в общем списке.

Нужно применить сумму с нарастающим итогом.

Допустим есть маленькая таблица:

	A	B	C	D
1	11		=A1	=СУММ(A1:A5)
2	22	=C1+1	=СУММ(A2;C1)	
3	33			
4	44			
5	55			

Условно столбец A - это количество человек, проживающих в одном из 5 регионов.

Нам нужно получить Первый номер в общем списке (столбец B) и Последний номер в общем списке (столбец C).

Сначала определимся с суммой всех проживающих в 5 регионах.

В ячейке D1 ставим `=СУММ(A1:A5)`, и нажимаем зелёную галочку или клавишу Enter. Получится 165. Это для проверки.

Столбец C отвечает у нас за сумму всех жителей с нарастающим итогом. Мы сначала копируем в ячейку C1 количество жителей в первом регионе. Для этого в ячейке C1 записываем формулу `=A1`. В следующей ячейке C2 записываем формулу `=СУММ(A2;C1)`. Нажимаем клавишу Enter. Появится уже необходимая нам сумма. Встаём на эту ячейку и за правый нижний угол, когда курсор примет вид крестика, протягиваем с зажатой левой кнопкой мыши до позиции C5. В ячейке C5 должна получиться такая же сумма, что и в D1, то есть 165.

Как уже говорили столбец C - Последний номер в общем списке.

Первые номера в общем списке (столбец B) получаем таким образом. В ячейку B1 записываем 1. Это старт нашей нумерации. В ячейку B2 записываем формулу `=C1+1`. Получаем номер, с которого начинается новый диапазон для данного региона. И протягиваем до конца столбца B2.

Шаг 1.

Действуя так же в отношении с таблицей регионов мы получим искомую таблицу со сквозными номерами жителей каждого региона. Только нужно учесть, что в этом случае первая строка содержит названия столбцов, поэтому адреса ячеек смещается на 1, то есть 2.

И столбец A у нас занимает информация о регионах.

Шаг 2.

Из этой таблицы возьмём 10 случайных чисел в интервале от 1 до 146447424 (столько у нас населения в России, и мы их всех пронумеровали по регионам в соответствии с их численностью)

Для нахождения случайных чисел можно воспользоваться генератором случайных чисел `=СЛУЧМЕЖДУ(1;146447424)` программы Excel.

Записываем эту формулу в любой свободный столбец. Получаем число из указанного диапазона. И эту ячейку протягиваем ещё на 9 позиций вниз. Надо помнить, что при повторном протягивании значения в ячейках будут меняться, так как это генератор случайных чисел.

Шаг 2.

По результатам Шага 2 выбираем регионы, в диапазоны которых попали номера случайных чисел.

Это можно сделать в ручном варианте, сравнивая полученные 10 случайных номеров с диапазонами каждого региона. В какой диапазон попал один из 10 номеров, такой регион и выбираем (10 номеров не такая уж большая выборка).

Шаг 3.

Из выбранных 10 регионов выбираем случайным образом 180 человек.

То есть, создаётся отдельная ячейка, в которую записывается формула `=СЛУЧМЕЖДУ(110400219;165114639379)`, если, например, первым из 10 регионов была Свердловская область. Нажимается Enter, левой кнопкой мыши протягивается нижний правый угол ячейки на 180 позиций. Список номеров в Свердловской области готов. Также поступаем с остальными 9 регионами.

Всего должно получиться 1800 человек (180×10).

Иногда может получиться в одной области несколько респондентов, так как мы делали выборку 10 номеров случайным образом из общей генеральной совокупности от 1 до 146447424. В таких случаях, регионы с двумя или тремя выбранными номерами респондентов увеличивают 180 на своё количество выбранных номеров. То есть, если попало на регион 2 номера респондентов, то будущая выборка по этому региону будет $180 \times 2 = 360$. Аналогично для 3 номеров в регионе: $180 \times 3 = 540$. Но это очень редкий случай. В любом варианте общее число выбранных респондентов из 10 регионов должно оказаться равным 1800. Надо заметить, что число регионов, выбранное случайным образом, и число человек из этих выбранных регионов даётся на откуп аналитику, так как любой аналитический расчёт требует тщательной подготовки данных.

А теперь предлагается решить эту задачу используя Python.

`python_excel.py`

```
from openpyxl import load_workbook
import random
workbook = load_workbook(filename="python_excel.xlsx")
workbook.sheetnames
sheet = workbook.active
max_row = sheet.max_row
max_column = sheet.max_column
name_column = ['Регион', 'Численность населения', 'Первый номер в общем списке', 'Последний номер в общем списке']
def n_column(max_column, name_column):
    dict_mame = {}
    for j in range(max_column):
        if sheet[1][j].value in name_column:
            dict_mame[sheet[1][j].value] = j + 1
    return dict_mame
dict_mame = n_column(max_column, name_column)
region = dict_mame['Регион']
population_size = dict_mame['Численность населения']
first_number = dict_mame['Первый номер в общем списке']
last_number = dict_mame['Последний номер в общем списке']
sheet[2][last_number - 1].value = sheet[2][population_size - 1].value
for row in range(max_row):
    if row < max_row - 3:
        sheet[row + 3][last_number - 1].value = sheet[row + 2][last_number - 1].value + sheet[row + 3][population_size - 1].value
sheet[2][first_number - 1].value = 1
for row in range(max_row):
    if row < max_row - 3:
        sheet[row + 3][first_number - 1].value = sheet[row + 2][last_number - 1].value + 1
```

```

workbook.save("python_excel.xlsx")
n_persons = []
for i in range(10):
    n_person = random.randrange(1, 146447424, 1)
    n_persons.append(n_person)
name_regions = []
for row in range(max_row):
    if row < max_row - 2:
        for n in range(10):
            if n_persons[n] >= sheet[row + 2][first_number - 1].value and
n_persons[n] < sheet[row + 2][last_number - 1].value:
                reg = sheet[row + 2][region - 1].value
                name_regions.append(reg)
dict_result = {}
for row in range(max_row):
    if row < max_row - 2:
        if sheet[row + 2][region - 1].value in name_regions:
            name_region = sheet[row + 2][region - 1].value
            cur_population_size = sheet[row + 2][population_size - 1].value
            cur_first_number = sheet[row + 2][first_number - 1].value
            cur_last_number = sheet[row + 2][last_number - 1].value
            dict_result[name_region] = [cur_population_size, cur_first_number,
cur_last_number]
results = []
for item in dict_result:
    #print(item, dict_result[item][1], dict_result[item][2])
    results.append(item)
    for j in range(180):
        result = random.randrange(dict_result[item][1], dict_result[item][2], 1)
        results.append(result)

import openpyxl
excel_doc = openpyxl.Workbook()
excel_doc.create_sheet(title = 'Лист1', index = 0)
sheetnames = excel_doc.sheetnames
sheet = excel_doc[sheetnames[0]]
i = 1
for row in results:
    sheet[f"A{i}"] = row
    i += 1
excel_doc.save('rezult.xlsx')

```

Файлы python_excel.py и python_excel.xlsx должны находиться в одном каталоге и чтобы запустить программу python_excel.py нужно в терминале записать:

`python(3) python_excel.py`

Напомним, что `python` означает запуск Python для Windows - `python`, для linux - `python3`.

Открываем Excel и видим, что наши расчёты были исполнены мгновенно, по сравнению с «ручным» вариантом. Кроме всего прочего, наша программа не зависит от данных. Например, если в таблице будут добавлены новые регионы России, то в «ручном» варианте придётся снова протягивать формулы, а уж, если вставляются новые столбы, то слетают все формулы.

Кроме всего прочего, наше приложение не только автоматически заполнило все пустые графы но и создала на выходе файл `rezult.xlsx`, в котором записаны все 1800 номеров, выбранных из генеральной совокупности в 146447424 жителя Российской Федерации. Подытоживая эту задачу, мы сначала записали начальный и конечный номер регионов по количеству жителей в том или ином регионе нарастающим итогом. Реализовали выборку 10 позиций случайным образом из 146447424. Выбирали регионы, в диапазонах которых попали выбранные числа. Далее, в выбранных регионах выбирали случайным образом по 180 номеров. В конечном счёте, полученная информация записывается в файл `rezult.xlsx`. Здесь будет записан список в виде одного столбца, в котором указывается имя региона, а ниже 180 (или 360) номеров жителей данного региона. Эти номера будут основой для анкетирования жителей выбранных регионов. По результатам анкетирования может быть проведён математический анализ, например, методом главных компонент, или факторный анализ.

Можно посмотреть количество и название формул, используемых в `openpyxl`.

```
from openpyxl.utils import FORMULAE
len(FORMULAE)
# Выведет: 352
FORMULAE
# Выведет:
frozenset({'ACCRINTM', 'TINV', 'VALUE', 'NORMSDIST', 'DEGREES', 'COSH', 'GAMMADIST',
'COUPDAYBS', 'CHOOSE', 'ACCRINT', 'PRICEMAT', 'CODE', 'COUPDAYS', 'BETAINV', 'MIN',
'AND', 'DOLLARDE', 'N', 'DEVSQ', 'AMORLINC', 'DEC2HEX', 'LOG', 'TANH', 'ISREF', 'FACT',
'MODE', 'STDEVP', 'ATANH', 'STDEVPSTEYX', 'MOD', 'HEX2DEC', 'RATE', 'PERMUT',
...
'CUBERANKEDMEMBER', 'SYD', 'ISEVEN', 'TRUNC', 'CUBESETCOUNT', 'LN', 'IMSIN', 'TAN',
'HYPERLINK', 'PEARSON', 'IMSUB', 'SUMX2PY2', 'EFFECT', 'LEN', 'IMEXP', 'VLOOKUP',
'ODDFPRICE', 'LOGINV', 'INT', 'DAYS360', 'SQRTPI'})
```

Создание игр на Python (библиотека PyGame)

На языке Python можно создавать различные по сложности игры. В частности библиотека PyGame позволяет разработать код для игр на плоскости. Pygame -

кроссплатформенна, бесплатна, с открытым исходным кодом, имеется хорошая документация, не требует установки дополнительных модулей для графики. Для установки на Windows можно выйти на сайт pygame.ru, скачать zip-файл, разархивировать его и запустить файл `pygame.exe` для установки и перезагрузить компьютер.

Для установки на linux в терминале запускается команда

```
sudo pip install pygame
```

или в некоторых дистрибутивах linux, в которых не установлена `sudo`, через `root` с дальнейшим выходом, то есть:

```
su
```

```
(Ввод пароля root)
```

```
pip install pygame
```

```
exit
```

При запуске программы `pip` сначала проверяется её установленная версия. Если версия старая, то система может предложить командную строку по установке новой версии, что-то вроде:

```
python3 -m pip install --upgrade pip
```

Ничего страшного, если у Вас уже была установлена программа Pygame. Это не мешает установить новый вариант программы, или никаких действий не произойдёт при наличии самой свежей версии.

Смысл ставить это приложение в отдельную виртуальную среду будет тогда, когда Вы планируете заниматься созданием игр в большом объёме, чтобы каждая игра, созданная Вами не конфликтовала с изменяющимися во времени версиями Python и PyGame.

Пример программы «Змейка»

Программа «Змейка», написанная с помощью библиотеки PyGame, взятой из открытых источников.

```
zmeyka.py
```

```
import pygame
```

```
import time
```

```
import random
```

```
pygame.init()
```

```
white = (255, 255, 255)
```

```
yellow = (255, 255, 102)
```

```
black = (0, 0, 0)
```

```
red = (213, 50, 80)
```

```
green = (0, 255, 0)
```

```
blue = (50, 153, 213)
```

```
dis_width = 800
```

```
dis_height = 600
```

```
dis = pygame.display.set_mode((dis_width, dis_height))
```

```
pygame.display.set_caption('Змейка от Skillbox')
```

```
clock = pygame.time.Clock()
```

```

snake_block = 10
snake_speed = 15
font_style = pygame.font.SysFont("bahnschrift", 25)
score_font = pygame.font.SysFont("comicsansms", 35)
def Your_score(score):
    value = score_font.render("Ваш счёт: " + str(score), True, yellow)
    dis.blit(value, [0, 0])

def our_snake(snake_block, snake_list):
    for x in snake_list:
        pygame.draw.rect(dis, black, [x[0], x[1], snake_block, snake_block])
def message(msg, color):
    mesg = font_style.render(msg, True, color)
    dis.blit(mesg, [dis_width / 6, dis_height / 3])
def gameLoop():
    game_over = False
    game_close = False
    x1 = dis_width / 2
    y1 = dis_height / 2
    x1_change = 0
    y1_change = 0
    snake_List = []
    Length_of_snake = 1
    foodx = round(random.randrange(0, dis_width - snake_block) / 10.0) * 10.0
    foody = round(random.randrange(0, dis_height - snake_block) / 10.0) * 10.0
    while not game_over:
        while game_close == True:
            dis.fill(blue)
            message("Вы проиграли! Нажмите Q для выхода или C для
повторной игры", red)
            Your_score(Length_of_snake - 1)
            pygame.display.update()
            for event in pygame.event.get():
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_q:
                        game_over = True
                        game_close = False
                    if event.key == pygame.K_c:
                        gameLoop()
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    game_over = True
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_LEFT:

```

```

        x1_change = -snake_block
        y1_change = 0
    elif event.key == pygame.K_RIGHT:
        x1_change = snake_block
        y1_change = 0
    elif event.key == pygame.K_UP:
        y1_change = -snake_block
        x1_change = 0
    elif event.key == pygame.K_DOWN:
        y1_change = snake_block
        x1_change = 0
    if x1 >= dis_width or x1 < 0 or y1 >= dis_height or y1 < 0:
        game_close = True
    x1 += x1_change
    y1 += y1_change
    dis.fill(blue)
    pygame.draw.rect(dis, green, [foodx, foody, snake_block, snake_block])
    snake_Head = []
    snake_Head.append(x1)
    snake_Head.append(y1)
    snake_List.append(snake_Head)
    if len(snake_List) > Length_of_snake:
        del snake_List[0]
    for x in snake_List[:-1]:
        if x == snake_Head:
            game_close = True
    our_snake(snake_block, snake_List)
    Your_score(Length_of_snake - 1)
    pygame.display.update()
    if x1 == foodx and y1 == foody:
        foodx = round(random.randrange(0, dis_width - snake_block) /
10.0) * 10.0
        foody = round(random.randrange(0, dis_height - snake_block) /
10.0) * 10.0
        Length_of_snake += 1
    clock.tick(snake_speed)
    pygame.quit()
    quit()
gameLoop()

```

Пример программы «Tetris»

Ещё одна известная игра: Tetris.

Для работы программы нужны библиотеки PyGame и NumPy. PyGame у нас уже установлен, а NumPy должен быть установлен, когда устанавливали Pandas с постфиксом [all]. Если же NumPy всё-таки не установлен, то набираем в терминале команду:

```
pip install numpy
```

```
tetris
```

```
"""
```

```
    The classic Tetris game developed using PyGame.
```

```
    Copyright (C) 2018 Recursos Python - recursospython.com.
```

```
    Copyright (C) 2021 Python Assets - pythonassets.com.
```

```
"""
```

```
from collections import OrderedDict
```

```
import random
```

```
from pygame import Rect
```

```
import pygame
```

```
import numpy as np
```

```
WINDOW_WIDTH, WINDOW_HEIGHT = 500, 601
```

```
GRID_WIDTH, GRID_HEIGHT = 300, 600
```

```
TILE_SIZE = 30
```

```
def remove_empty_columns(arr, _x_offset=0, _keep_counting=True):
```

```
    """
```

```
    Remove empty columns from arr (i.e., those filled with zeros).
```

```
    The return value is (new_arr, x_offset), where x_offset is how  
    much the x coordinate needs to be increased in order to maintain  
    the block's original position.
```

```
    """
```

```
    for colid, col in enumerate(arr.T):
```

```
        if col.max() == 0:
```

```
            if _keep_counting:
```

```
                _x_offset += 1
```

```
            # Remove the current column and try again.
```

```
            arr, _x_offset = remove_empty_columns(
```

```
                np.delete(arr, colid, 1), _x_offset, _keep_counting)
```

```
            break
```

```
        else:
```

```
            _keep_counting = False
```

```
    return arr, _x_offset
```

```
class BottomReached(Exception):
```

```
    pass
```

```
class TopReached(Exception):
```

```
    pass
```

```
class Block(pygame.sprite.Sprite):
```

```
    @staticmethod
```

```
    def collide(block, group):
```

```

"""
Check if the specified block collides with some other block
in the group.
"""
for other_block in group:
    # Ignore the current block which will always collide with itself.
    if block == other_block:
        continue
    if pygame.sprite.collide_mask(block, other_block) is not None:
        return True
return False
def __init__(self):
    super().__init__()
    # Get a random color.
    self.color = random.choice((
        (200, 200, 200),
        (215, 133, 133),
        (30, 145, 255),
        (0, 170, 0),
        (180, 0, 140),
        (200, 200, 0)
    ))
    self.current = True
    self.struct = np.array(self.struct)
    # Initial random rotation and flip.
    if random.randint(0, 1):
        self.struct = np.rot90(self.struct)
    if random.randint(0, 1):
        # Flip in the X axis.
        self.struct = np.flip(self.struct, 0)
    self._draw()
def _draw(self, x=4, y=0):
    width = len(self.struct[0]) * TILE_SIZE
    height = len(self.struct) * TILE_SIZE
    self.image = pygame.surface.Surface([width, height])
    self.image.set_colorkey((0, 0, 0))
    # Position and size
    self.rect = Rect(0, 0, width, height)
    self.x = x
    self.y = y
    for y, row in enumerate(self.struct):
        for x, col in enumerate(row):
            if col:
                pygame.draw.rect(

```

```

        self.image,
        self.color,
        Rect(x*TILE_SIZE + 1, y*TILE_SIZE + 1,
            TILE_SIZE - 2, TILE_SIZE - 2)
    )
    self._create_mask()
def redraw(self):
    self._draw(self.x, self.y)
def _create_mask(self):
    """
    Create the mask attribute from the main surface.
    The mask is required to check collisions. This should be called
    after the surface is created or update.
    """
    self.mask = pygame.mask.from_surface(self.image)
def initial_draw(self):
    raise NotImplementedError
@property
def group(self):
    return self.groups()[0]
@property
def x(self):
    return self._x
@x.setter
def x(self, value):
    self._x = value
    self.rect.left = value*TILE_SIZE
@property
def y(self):
    return self._y
@y.setter
def y(self, value):
    self._y = value
    self.rect.top = value*TILE_SIZE

def move_left(self, group):
    self.x -= 1
    # Check if we reached the left margin.
    if self.x < 0 or Block.collide(self, group):
        self.x += 1
def move_right(self, group):
    self.x += 1
    # Check if we reached the right margin or collided with another
    # block.

```

```
if self.rect.right > GRID_WIDTH or Block.collide(self, group):
    # Rollback.
    self.x -= 1
```

```
def move_down(self, group):
    self.y += 1
    # Check if the block reached the bottom or collided with
    # another one.
    if self.rect.bottom > GRID_HEIGHT or Block.collide(self, group):
        # Rollback to the previous position.
        self.y -= 1
        self.current = False
        raise BottomReached
```

```
def rotate(self, group):
    self.image = pygame.transform.rotate(self.image, 90)
    # Once rotated we need to update the size and position.
    self.rect.width = self.image.get_width()
    self.rect.height = self.image.get_height()
    self._create_mask()
    # Check the new position doesn't exceed the limits or collide
    # with other blocks and adjust it if necessary.
    while self.rect.right > GRID_WIDTH:
        self.x -= 1
    while self.rect.left < 0:
        self.x += 1
    while self.rect.bottom > GRID_HEIGHT:
        self.y -= 1
    while True:
        if not Block.collide(self, group):
            break
        self.y -= 1
    self.struct = np.rot90(self.struct)
```

```
def update(self):
    if self.current:
        self.move_down()
```

```
class SquareBlock(Block):
    struct = (
        (1, 1),
        (1, 1)
    )
```

```
class TBlock(Block):
    struct = (
        (1, 1, 1),
```

```

        (0, 1, 0)
    )
class LineBlock(Block):
    struct = (
        (1,),
        (1,),
        (1,),
        (1,)
    )
class LBlock(Block):
    struct = (
        (1, 1),
        (1, 0),
        (1, 0),
    )
class ZBlock(Block):
    struct = (
        (0, 1),
        (1, 1),
        (1, 0),
    )
class BlocksGroup(pygame.sprite.OrderedUpdates):
    @staticmethod
    def get_random_block():
        return random.choice(
            (SquareBlock, TBlock, LineBlock, LBlock, ZBlock))()
    def __init__(self, *args, **kwargs):
        super().__init__(self, *args, **kwargs)
        self._reset_grid()
        self._ignore_next_stop = False
        self.score = 0
        self.next_block = None
        # Not really moving, just to initialize the attribute.
        self.stop_moving_current_block()
        # The first block.
        self._create_new_block()
    def _check_line_completion(self):
        """
        Check each line of the grid and remove the ones that
        are complete.
        """
        # Start checking from the bottom.
        for i, row in enumerate(self.grid[::-1]):
            if all(row):

```

```

self.score += 5
# Get the blocks affected by the line deletion and
# remove duplicates.
affected_blocks = list(
    OrderedDict.fromkeys(self.grid[-1 - i]))
for block, y_offset in affected_blocks:
    # Remove the block tiles which belong to the
    # completed line.
    block.struct = np.delete(block.struct, y_offset, 0)
    if block.struct.any():
        # Once removed, check if we have empty
columns
        # since they need to be dropped.
        block.struct, x_offset =
remove_empty_columns(block.struct)
        # Compensate the space gone with the columns
to
        # keep the block's original position.
        block.x += x_offset
        # Force update.
        block.redraw()
    else:
        # If the struct is empty then the block is gone.
        self.remove(block)
# Instead of checking which blocks need to be moved
# once a line was completed, just try to move all of
# them.
for block in self:
    # Except the current block.
    if block.current:
        continue
    # Pull down each block until it reaches the
    # bottom or collides with another block.
    while True:
        try:
            block.move_down(self)
        except BottomReached:
            break
self.update_grid()
# Since we've updated the grid, now the i counter
# is no longer valid, so call the function again
# to check if there're other completed lines in the
# new grid.
self._check_line_completion()

```

```

        break
def _reset_grid(self):
    self.grid = [[0 for _ in range(10)] for _ in range(20)]
def _create_new_block(self):
    new_block = self.next_block or BlocksGroup.get_random_block()
    if Block.collide(new_block, self):
        raise TopReached
    self.add(new_block)
    self.next_block = BlocksGroup.get_random_block()
    self.update_grid()
    self._check_line_completion()
def update_grid(self):
    self._reset_grid()
    for block in self:
        for y_offset, row in enumerate(block.struct):
            for x_offset, digit in enumerate(row):
                # Prevent replacing previous blocks.
                if digit == 0:
                    continue
                rowid = block.y + y_offset
                colid = block.x + x_offset
                self.grid[rowid][colid] = (block, y_offset)

@property
def current_block(self):
    return self.sprites()[-1]
def update_current_block(self):
    try:
        self.current_block.move_down(self)
    except BottomReached:
        self.stop_moving_current_block()
        self._create_new_block()
    else:
        self.update_grid()
def move_current_block(self):
    # First check if there's something to move.
    if self._current_block_movement_heading is None:
        return
    action = {
        pygame.K_DOWN: self.current_block.move_down,
        pygame.K_LEFT: self.current_block.move_left,
        pygame.K_RIGHT: self.current_block.move_right
    }
    try:
        # Each function requires the group as the first argument

```

```

        # to check any possible collision.
        action[self._current_block_movement_heading](self)
    except BottomReached:
        self.stop_moving_current_block()
        self._create_new_block()
    else:
        self.update_grid()
def start_moving_current_block(self, key):
    if self._current_block_movement_heading is not None:
        self._ignore_next_stop = True
        self._current_block_movement_heading = key
def stop_moving_current_block(self):
    if self._ignore_next_stop:
        self._ignore_next_stop = False
    else:
        self._current_block_movement_heading = None
def rotate_current_block(self):
    # Prevent SquareBlocks rotation.
    if not isinstance(self.current_block, SquareBlock):
        self.current_block.rotate(self)
        self.update_grid()
def draw_grid(background):
    """Draw the background grid."""
    grid_color = 50, 50, 50
    # Vertical lines.
    for i in range(11):
        x = TILE_SIZE * i
        pygame.draw.line(
            background, grid_color, (x, 0), (x, GRID_HEIGHT)
        )
    # Horizontal liens.
    for i in range(21):
        y = TILE_SIZE * i
        pygame.draw.line(
            background, grid_color, (0, y), (GRID_WIDTH, y)
        )

def draw_centered_surface(screen, surface, y):
    screen.blit(surface, (400 - surface.get_width()//2, y))
def main():
    pygame.init()
    pygame.display.set_caption("Tetris with PyGame")
    screen = pygame.display.set_mode((WINDOW_WIDTH, WINDOW_HEIGHT))

```

```

run = True
paused = False
game_over = False
# Create background.
background = pygame.Surface(screen.get_size())
bgcolor = (0, 0, 0)
background.fill(bgcolor)
# Draw the grid on top of the background.
draw_grid(background)
# This makes blitting faster.
background = background.convert()
try:
    font = pygame.font.Font("Roboto-Regular.ttf", 20)
except OSError:
    # If the font file is not available, the default will be used.
    font = pygame.font.Font(pygame.font.get_default_font(), 20)
next_block_text = font.render(
    "Next figure:", True, (255, 255, 255), bgcolor)
score_msg_text = font.render(
    "Score:", True, (255, 255, 255), bgcolor)
game_over_text = font.render(
    "!Game over!", True, (255, 220, 0), bgcolor)
# Event constants.
MOVEMENT_KEYS = pygame.K_LEFT, pygame.K_RIGHT, pygame.K_DOWN
EVENT_UPDATE_CURRENT_BLOCK = pygame.USEREVENT + 1
EVENT_MOVE_CURRENT_BLOCK = pygame.USEREVENT + 2
pygame.time.set_timer(EVENT_UPDATE_CURRENT_BLOCK, 1000)
pygame.time.set_timer(EVENT_MOVE_CURRENT_BLOCK, 100)
blocks = BlocksGroup()
while run:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False
            break
        elif event.type == pygame.KEYUP:
            if not paused and not game_over:
                if event.key in MOVEMENT_KEYS:
                    blocks.stop_moving_current_block()
                elif event.key == pygame.K_UP:
                    blocks.rotate_current_block()
            if event.key == pygame.K_p:
                paused = not paused
    # Stop moving blocks if the game is over or paused.
    if game_over or paused:

```

```

        continue
    if event.type == pygame.KEYDOWN:
        if event.key in MOVEMENT_KEYS:
            blocks.start_moving_current_block(event.key)
    try:
        if event.type == EVENT_UPDATE_CURRENT_BLOCK:
            blocks.update_current_block()
        elif event.type == EVENT_MOVE_CURRENT_BLOCK:
            blocks.move_current_block()
    except TopReached:
        game_over = True

    # Draw background and grid.
    screen.blit(background, (0, 0))
    # Blocks.
    blocks.draw(screen)
    # Sidebar with misc. information.
    draw_centered_surface(screen, next_block_text, 50)
    draw_centered_surface(screen, blocks.next_block.image, 100)
    draw_centered_surface(screen, score_msg_text, 240)
    score_text = font.render(
        str(blocks.score), True, (255, 255, 255), bgcolor)
    draw_centered_surface(screen, score_text, 270)
    if game_over:
        draw_centered_surface(screen, game_over_text, 360)
    # Update.
    pygame.display.flip()
    pygame.quit()
if __name__ == "__main__":
    main()

```

Создание сайтов: фреймворк Django

Все файлы и каталоги для этого раздела находятся по адресу:
pageforweb.ru/django_dir.zip

Попробуем создать фрагмент сайта спортивных секций. В нём будет информация только о тренерах каждой спортивной секции. В более общем варианте сайт, помимо информации о тренерах, будет включать в себя перечень спортсменов, записавшихся в ту или иную секцию, графики занятий и соревнований, обсуждение результатов, продажа экипировки и инвентаря и некоторые другие возможности.

Без подробного объяснения покажем шаги по созданию такого фрагмента сайта. Устанавливаем фреймворк Django. Лучше его установить в отдельную виртуальную среду. Напоминаем, что модуль поддерживает создание

облегченных виртуальных сред `venv` имеет свой собственный независимый набор пакетов Python.

Создаём новый каталог для Windows графической утилитой:

```
C:\>django
```

Для linux

```
mkdir /home/<Ваш домашний каталог>/django
```

Перейдём в новый каталог.

```
cd /home/<Ваш домашний каталог>/django
```

Установим виртуальную среду.

```
python3 -m venv sport
```

Проверим.

```
ls -la
```

Активация виртуального окружения выполняется по-разному для Windows и Linux соответственно:

```
sport\Scripts\activate.bat
```

```
source sport/bin/activate
```

Если всё сделано правильно, то появиться сообщение, что-то вроде:

```
(sport) [<Ваш домашний каталог>@localhost django]$
```

```
ls
```

```
bin include lib lib64 pyvenv.cfg sports
```

Теперь можно установить фреймворк Django, вначале перейдя на нужный каталог, в котором мы создавали виртуальную среду, и набрать в терминале после активации виртуальной среды:

```
(sport) [<Ваш домашний каталог>@localhost django]$ python -m pip install Django
```

В дальнейшем приглашение для ввода командной строки `(sport) [<Ваш домашний каталог>@localhost django]$` будем пропускать, а писать только то, что будет вводиться с клавиатуры.

После установки Django запускаем проект `sports`:

```
django-admin startproject sports
```

Перейдём в образованный каталог `sports`.

```
cd sports
```

Посмотрим, что в этом каталоге

```
ls
```

Вот какие каталоги и файлы появятся у нас после создания проекта `sports`:

```
db.sqlite3 manage.py sports
```

Здесь `db.sqlite3` это файл с базой данных SQLite3. Как видно она автоматически образовалась при установке Django. Дальнейшее использование этой базы из Django, предусматривает соединение именно по такому имени базы данных. Файл `manage.py` - управление проектом.

Каталог `sports`, идентичный по названию проекта, имеет уровень приложения. Он имеет первоначально в своём составе следующие файлы:

```
asgi.py __init__.py __pycache__ settings.py urls.py wsgi.py
```

Пока не будем описывать назначение этих файлов, а ещё раз уточним, что имеются два уровня: проекта и приложения, которые имеют одинаковые названия `sports`. Уровень приложения вложен в уровень проекта.

После всех установок, сделаем себе хорошее настроение, запустим наш новый сервер с уровня проекта. Хороший ориентир, показывающий, что мы находимся на уровне проекта это наличие файла управления проектом `manage.py`.

`python3 manage.py runserver`

Будет выведено сообщение о запуске веб-сервера, с адресом веб-браузера и указанием как можно остановить веб-сервер, нажав `Ctrl+Z`.

Переходим в веб-браузер и набираем адрес:

<http://127.0.0.1:8000/>

Видим приветственную картинку Django. Первый шаг сделан.

django

Просмотреть [примечания к выпуску Django](#)



Установка прошла успешно! Поздравляем!

Вы видите эту страницу, потому что в вашем файле настроек указано `DEBUG=True` и вы не настроили никаких URL-адресов.



Документация по Django

Темы, ссылки и практические рекомендации



Учебное пособие:
приложение для опроса
Начало работы с Django



Сообщество Django

Подключайтесь, получайте помощь или вносите свой вклад

Это уже переведённая картинка. В начале она на английском. Работа на русском языке устанавливается в настройках. Но всё по порядку. Остановим веб-сервер, набрав на терминале `Ctrl+C`. Снова взглянем на получившуюся иерархию проекта `sports`.

sports

`db.sqlite3`

`manage.py`

sports

`asgi.py`

`__init__.py`

`__pycache__`

`settings.py`

`urls.py`

wsgi.py

Жирным шрифтом обозначены каталоги. Внутренний каталог, обозначенный синим, имеет такое же имя - sports, что и внешний каталог, обозначенный чёрным. Файлы db.sqlite3, manage.py и каталог sports со всеми вложенными в него файлами, обозначенными зелёным цветом, относятся к уровню проекта. Шаг 1.

Создаём приложение sports_coaches в терминале. Для этого встаём на уровень проекта (ориентируемся по наличию в папке проекта программы manage.py):

`python3 manage.py startapp sports_coaches`

У нас на уровне проекта образовалась папка с соответствующим запросу именем sports_coaches со следующими файлами и каталогами:

`admin.py`
`apps.py`
`__init__.py`
`migrations`
`models.py`
`tests.py`
`views.py`

Опять же не будем их расшифровывать.

Папка sports_coaches, со всеми входящими в него файлами и каталогами, будет относиться к уровню приложения.

Не выходя с уровня проекта отредактируем файл установки settings.py.

В список инсталлированных приложений добавим анонсированное приложение sports_coaches. Исходный текст:

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
]
```

Должно получиться:

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'sports_coaches.apps.SportsCoachesConfig', # Добавляется наше приложение  
]
```

Все остальные приложения системные.

База данных описывается фразой:

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.sqlite3',  
        'NAME': BASE_DIR / 'db.sqlite3',  
    }  
}
```

Оставляем не тронутой, что подразумевает использование в качестве системы управления базами данных по умолчанию однофайловую базу данных SQLite с именем db.sqlite3, находящийся в директории нашего проекта.

Ещё пару настроек в файле settings.py.

Оставляем настройку `DEBUG = True`, чтобы отслеживать ошибки в отладочном режиме. При установке же на сервер, для промышленной эксплуатации, этот параметр устанавливается в `DEBUG = False`.

Меняем `LANGUAGE_CODE = 'en-us'` на `LANGUAGE_CODE = 'ru-ru'`, чтобы создание проекта сайта велось на русском языке.

Пока, что мы создали и зарегистрировали пустое приложение `sports_coaches`.

Для него в папке apps, уровня приложения уже будет прописан класс `SportsCoachesConfig`. Но собственно приложения пока нет. Приложение прописывается в файле `views.py`. Обычно называют это представление, но по сути это есть контроллер нашего приложения. Поэтому часто его называют контроллером.

Попробуем оживить этот файл. Запишем туда нехитрый ответ веб-сервера на запрос пользователя, например, к страничке сайта `index.html`.

```
from django.http import HttpResponse  
def index(request) :  
    return HttpResponse("Главная страница")
```

Функция `index()` в файле `views.py` - контроллер. Но с этим контроллером не связан интернет-адрес. Вот для этого существуют маршруты и маршрутизаторы. Если мы свяжем контроллер, а точнее функцию `index()`, с маршрутом, направленным от интернет-адреса к исполнительной функции контроллера.

Интернет-адрес должен иметь определенный формат. Его называют шаблонный интернет-адрес. Шаблонный интернет-адрес не должен начинаться со слеша, не иметь названия протокола, адреса хоста, номера порта и GET-параметров.

Конечный слеш обязателен. Список шаблонных адресов прописывается в файле `urls.py` уровня проекта.

Вначале файл `urls.py` уровня проекта имеет вид:

```
from django.contrib import admin  
from django.urls import path  
urlpatterns = [  
    path('admin/', admin.site.urls),  
]
```

Такое содержание этого файла создается автоматически когда мы инициировали приложение `sports_coaches`. Django имеет уже

предустановленный блок управления сайтом через страницу администрирования admin/. К адресу администрирования сайта прибавляется шаблонный адрес нашего приложения sports_coaches.

```
from django.contrib import admin
from django.urls import path
from sports_coaches.views import index
urlpatterns = [
    path ( "", index),
    path ( 'admin/', admin. site. urls),
]
```

Теперь можно посмотреть, что получилось с нашим сайтом. Сначала активизируем его из папки уровня проекта набрав в терминале:

```
python3 manage.py runserver
```

В веб-браузере перейдём на страницу с адресом:

<http://localhost:8000/>

Попробуем сделать ещё несколько таких страниц-заглушек. Напишем для них функции в файле views.py уровня приложения и определим пути к ним в файле urls.py уровня проекта.

Преобразуем файл views.py следующим образом:

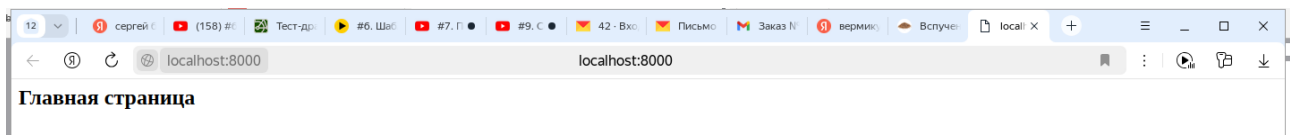
```
from django.http import HttpResponse
def index(request) :
    return HttpResponse("Главная страница")
def sports_coaches(request) :
    return HttpResponse("Страница приложения тренеров по различным видам спорта — sports_coaches.")
def about(request) :
    return HttpResponse("Информация о нас")
```

Модифицируем также файл urls.py уровня проекта:

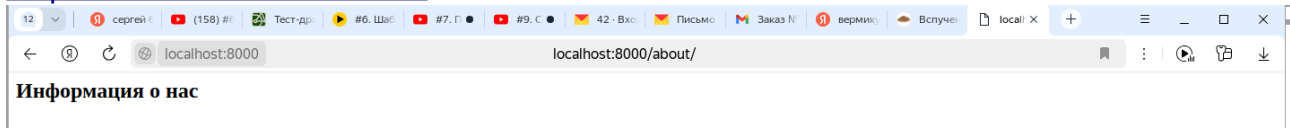
```
from django.contrib import admin
from django.urls import path
from sports_coaches.views import index, about, sports_coaches
urlpatterns = [
    path ( "", index),
    path ( 'sports_coaches/', sports_coaches),
    path ( 'about/', about),
    path ( 'admin/', admin. site. urls),
]
```

Получим следующие картинки при заходе по адресам:

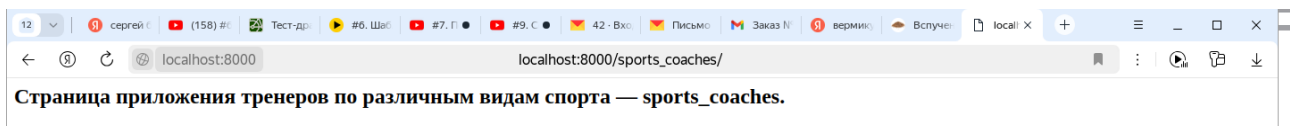
<http://localhost:8000/>



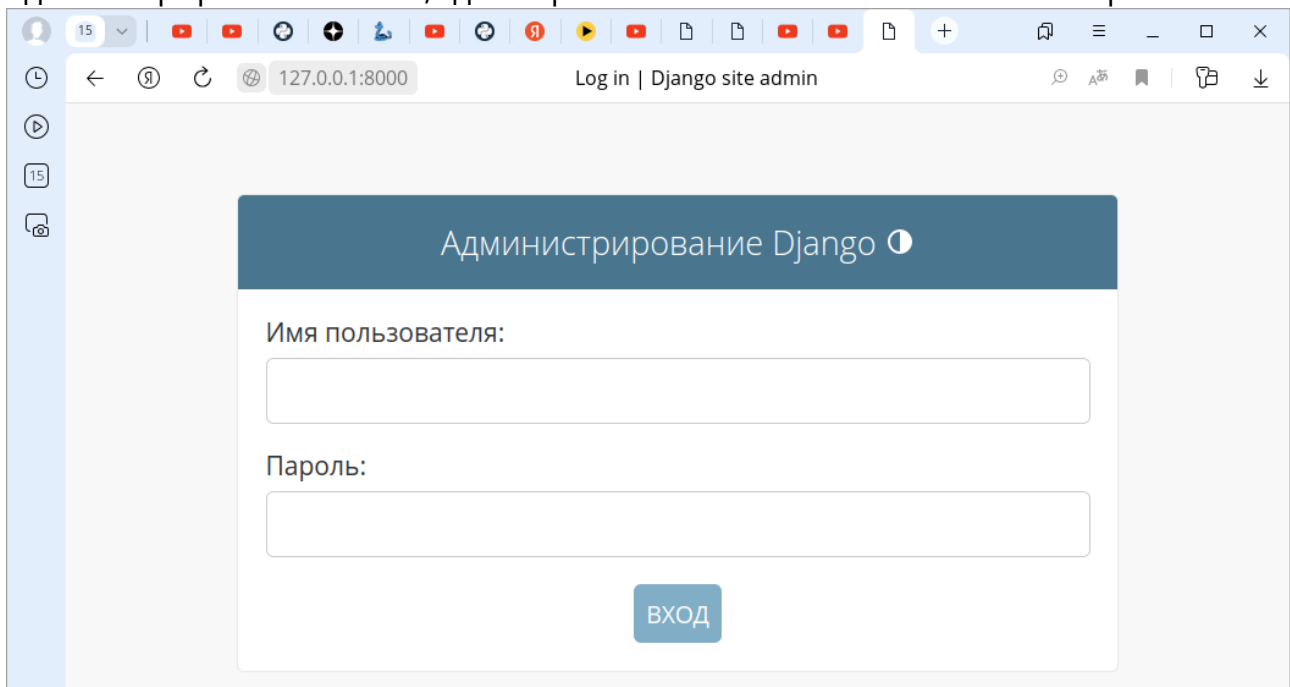
<http://localhost:8000/about/>



http://localhost:8000/sports_coaches/



А, если мы введём адрес <http://localhost:8000/admin/>, то выйдем на страничку администрирования сайта, где запрашивается имя пользователя и пароль.



Пока на сайте никаких пользователей нет. Активизируем главного пользователя - суперпользователя сайта.

`python3 manage.py createsuperuser`

В диалоге будет затребовано имя суперпользователя. По умолчанию (то есть не нужно ничего вводить) имя суперпользователя будет такое же как имя, под которым Вы входили с систему. Ещё будут затребованы e-mail для суперпользователя и пароль. Пароль должен содержать и буквы латинского алфавита и цифры и иметь не менее 8 символов. После подтверждения пароля будет создан суперпользователь веб-сайта. Проверим это.

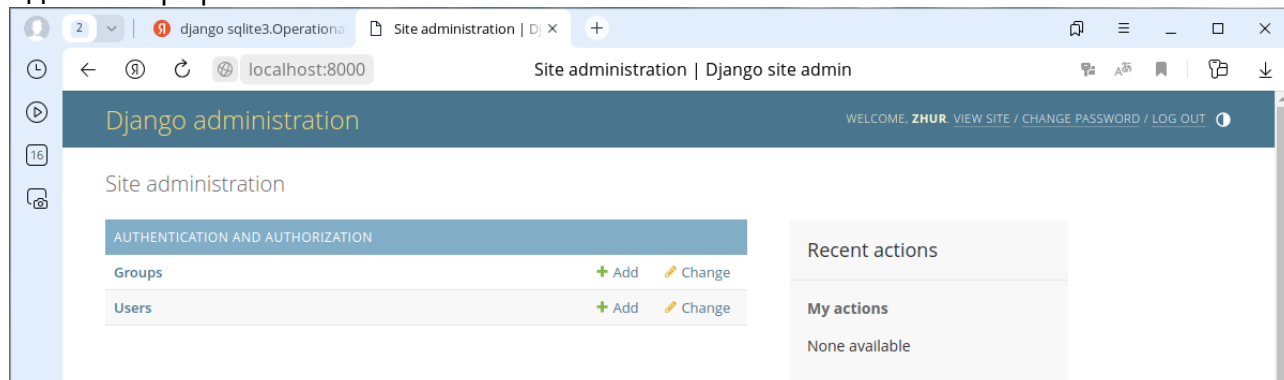
Снова запускаем веб-сервер:

`python3 manage.py runserver`

И в браузере введём адрес:

<http://localhost:8000/admin/>

Заполним поля имени пользователя и пароля и попадём на страничку администрирования веб-сайта.



Пока раздел администрирования малоинформативен. В дальнейшем мы это исправим.

А сейчас перейдём к созданию так называемых моделей. Они представляют из себя классы, описывающие содержание, взаимодействия и поведения данных нашего приложения. На уровне приложения находим файл `models.py` и модифицируем его, записав в него следующий текст:

```
from django.db import models
class SportsCoaches(models.Model):
    sport_coache = models.CharField(max_length=250)
    info = models.TextField()
    photo = models.ImageField(upload_to='media/')
    def __str__(self):
        return self.sport_coache
```

Посмотрим какие поля и их типы присутствуют в описании класса `SportsCoaches`:

1. `sport_coache` - фамилия имя отчество спортивного тренера. Тип поля символьный `CharField` с максимальной размерностью строки 250 символов.
2. `info` - информация о конкретном спортивном тренере. Его краткая биография и спортивные достижения. Тип поля также символьный, но с большой размерностью - `TextField`.
3. `ImageField` - фоторграфия конкретного спортивного тренера. Тип поля - `ImageField`. Этот тип поля характеризуется наличием файла с фотографией. Сами фотографии не хранятся в модели. Там прописываются только ссылки на файлы с фотографиями. Чтобы эти ссылки срабатывали и Django смог вывести эти фотографии, нужно определить подкаталог, где будут храниться загруженные изображения. Для этого в файл `settings.py` уровня приложения нужно прописать следующие команды:

```
import os
```

```
MEDIA_URL = "/media/"
```

```
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

MEDIA_URL - URL-адрес для медиафайлов и указываем корневой путь, по которому будет сохранен файл MEDIA_ROOT.

Создадим каталог media на уровне проекта (ориентир - файл manage.py) и помещаем туда подготовленные файлы с фотографиями спортивных тренеров. В качестве тренеров будем использовать информацию выдающихся спортсменов России по различным видам спорта. Информация о них взята из интернета. Нужно применить все миграции путём написания следующей команды:

```
python3 manage.py makemigrations
```

Выполнение этой команды может дать ошибку:

ERRORS:

sports_coaches.SportsCoaches.photo: (fields.E210) Cannot use ImageField because Pillow is not installed.

HINT: Get Pillow at <https://pypi.org/project/Pillow/> or run command "python -m pip install Pillow".

Это означает, что не установлена библиотека обработки изображений Pillow в данной виртуальной среде (sport) и её можно установить командой:

```
python -m pip install Pillow
```

После устранения ошибки снова запускаем команду создания миграционной модели:

```
python3 manage.py makemigrations
```

А затем проводим и саму миграцию:

```
python3 manage.py migrate
```

Будут выведено:

Operations to perform:

Apply all migrations: admin, auth, contenttypes, sessions

Running migrations:

Applying contenttypes.0001_initial... OK

Applying auth.0001_initial... OK

Applying admin.0001_initial... OK

Applying admin.0002_logentry_remove_auto_add... OK

Applying admin.0003_logentry_add_action_flag_choices... OK

Applying contenttypes.0002_remove_content_type_name... OK

Applying auth.0002_alter_permission_name_max_length... OK

Applying auth.0003_alter_user_email_max_length... OK

Applying auth.0004_alter_user_username_opts... OK

Applying auth.0005_alter_user_last_login_null... OK

Applying auth.0006_require_contenttypes_0002... OK

Applying auth.0007_alter_validators_add_error_messages... OK

Applying auth.0008_alter_user_username_max_length... OK

Applying auth.0009_alter_user_last_name_max_length... OK

Applying auth.0010_alter_group_name_max_length... OK

Applying auth.0011_update_proxy_permissions... OK

```
Applying auth.0012_alter_user_first_name_max_length... OK
Applying sessions.0001_initial... OK
```

Миграции создадут актуальные таблицы в базе данных.

Содержание файла 0001_initial.py выглядит следующим образом:

```
# Generated by Django 4.2.13 on 2024-07-26 17:17
```

```
from django.db import migrations, models
```

```
class Migration(migrations.Migration):
```

```
    initial = True
```

```
    dependencies = [
    ]
```

```
    operations = [
        migrations.CreateModel(
            name='SportsCoaches',
            fields=[
                ('id', models.BigAutoField(auto_created=True, primary_key=True,
serialize=False, verbose_name='ID')),
                ('sport_coache', models.CharField(max_length=250)),
                ('info', models.TextField()),
                ('photo', models.ImageField(upload_to='uploads/')),
            ],
        ),
    ]
```

Зарегистрируем нашу модель в admin.py, который находится на уровне приложения. Это необходимо для того, чтобы в админ сайте появился новая модель, которую в админ-сайте легко заполнять и изменять, а также удалять ненужную информацию. В админ-сайте всё уже подготовлено для редактирования данных в моделях. Открываем файл admin.py уровня приложения и регистрируем нашу модель. Первоначально файл admin.py имеет следующее содержание:

```
from django.contrib import admin
```

```
# Register your models here.
```

Изменяем его добавлением регистрации нашей модели:

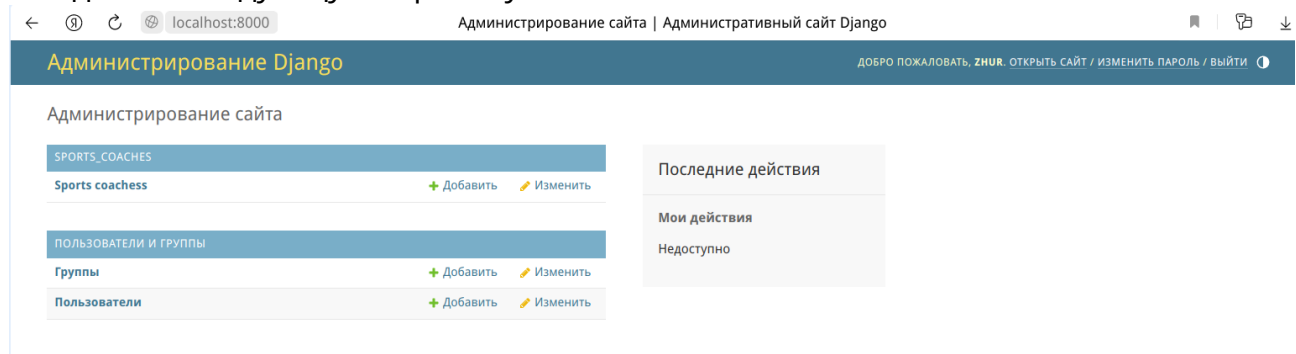
```
from django.contrib import admin
```

```
# Register your models here.  
from .models import SportsCoaches
```

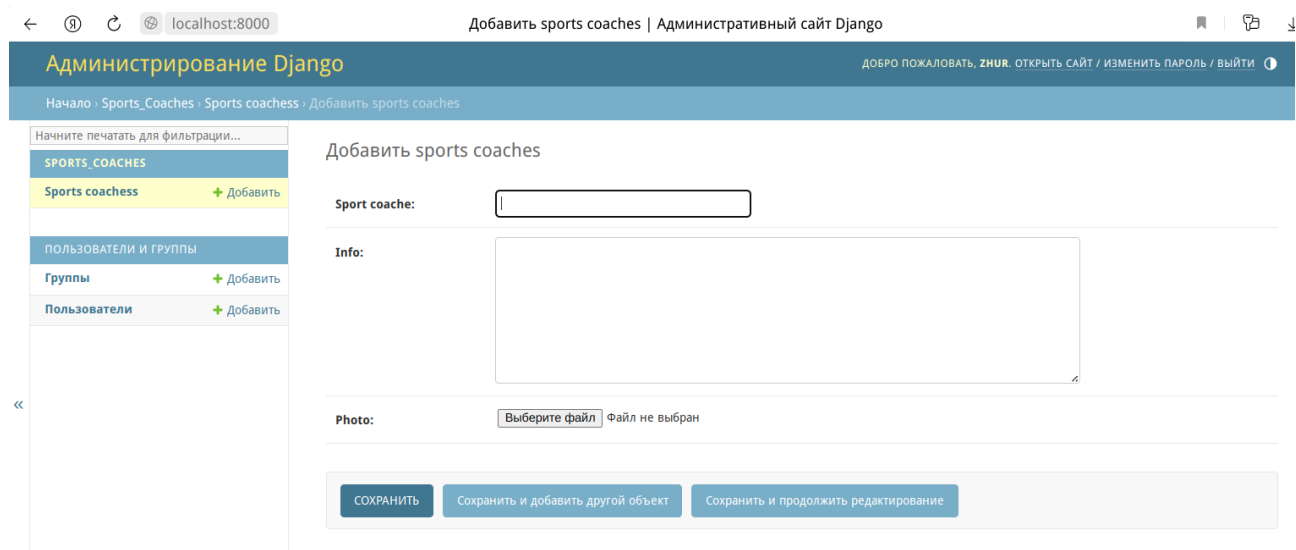
```
admin.site.register(SportsCoaches)
```

Запомнив этот файл, переходим на уровень проекта, и запускаем веб-сервер:
`python3 manage.py runserver`

Перейдя по ссылке <http://localhost:8000/admin/> и заполнив логин и пароль администратора сайта (регистрацию администратора сайта мы уже прошли), выйдем на следующую картинку:



Как видим наша модель была добавлена. Теперь можно переходить к редактированию данных нашей модели. Нажмём на ссылку +Добавить модели Sports couches и начнём редактирование данных в открывшейся форме:



В поле Sport coachess вставим имя и фамилию тренера:

Александр Большунов - лыжи

В поле Info вставляем информацию о биографии и спортивных достижениях тренера:

Александр Александрович Большунов (род. 31 декабря 1996, Подывожье, Севский район, Брянская область, Россия[2]) — российский лыжник, трехкратный олимпийский чемпион (2022), чемпион мира в скиатлоне (2021), двукратный обладатель Кубка мира (2019/20, 2020/21), двукратный победитель лыжной многодневной гонки Тур де Ски (2019/2020 и 2021). Двукратный победитель и

серебряный призёр чемпионатов мира по бегу на роликовых лыжах. Заслуженный мастер спорта России (2018). Универсал, успешно выступающий как в спринтерских, так и в дистанционных гонках.

Первый и единственный среди мужчин-лыжников, завоевавший олимпийские медали во всех шести олимпийских дисциплинах, а также наибольшее их число на одной зимней Олимпиаде — 5. Рекордсмен среди всех мужчин в истории советского и российского спорта по общему количеству медалей зимних Олимпийских игр — 9. Завоевывал медали во всех олимпийских гонках, в которых принимал участие, без исключения.

В поле Photo вставляем файл:

[aleksandr-bolshunov.jpg](#)

Напоминаем, что файлы с фотографиями были загружены в каталог media уровня приложения, где находится файл manage.py.

Далее, аналогично загрузим информацию о других тренерах.

Поле Sport coach:

[Юлия Ефимова - плавание](#)

Поле Info:

Юлия Андреевна Ефимова — российская пловчиха, заслуженный мастер спорта. Родилась 3 апреля 1992 года в Грозном.

Плаванием начала заниматься с шести лет в СДЮШОР № 2 в Волгодонске Ростовской области. Первым тренером Юлии был её отец Андрей Ефимов.

С 2006 года — член юниорской сборной России по плаванию, с 2007 года — член сборной команды России по плаванию.

Ефимова — трёхкратная чемпионка Европы 2007 года на короткой воде, трёхкратная чемпионка Европы на длинной воде (2008, 2010). Неоднократная победительница этапов Кубка мира. Многократная чемпионка (2008–2010) и рекордсменка России.

В 2008 году Всероссийская федерация плавания признала Юлию Ефимову лучшей пловчихой страны и «Открытием года».

Ефимова награждена медалью ордена «За заслуги перед Отечеством» II степени (2012) и почётной грамотой президента РФ (2013 года) за высокие спортивные достижения на XXVII Всемирной летней универсиаде 2013 года в городе Казани.

В поле Photo загружаем файл:

[julia-efimova.jpeg](#)

Поле Sport coach:

[Антон Чупков - плавание](#)

Поле Info:

Антон Михайлович Чупков (род. 22 февраля 1997, Москва) — российский пловец, двукратный чемпион мира, двукратный чемпион Европы, бронзовый призёр Олимпийских игр 2016 года на дистанции 200 метров брассом[2], заслуженный мастер спорта России[3].

Многократный обладатель юношеских рекордов мира[англ.], победитель и призёр Летних юношеских Олимпийских игр 2014 года в Нанкине[4], четырёхкратный чемпион юниорского первенства мира по плаванию[англ.][5], четырёхкратный чемпион I Европейских игр 2015 в Баку, чемпион Всемирных военных игр[6], многократный рекордсмен, победитель и призёр первенств России и Европы среди юниоров, чемпион России на дистанции 200 м брассом.

В поле Photo загружаем файл:

anton-cvhupkov.jpeg

Поле Sport coach:

Саян Хертек - самбо

Поле Info:

Саян Калдар-оолович Хертек (род. 5 сентября 1987) — российский самбист и дзюдоист. Трёхкратный чемпион мира по самбо (Сочи-2017, Бишкек-2022, Ереван-2023), многократный чемпион России по самбо, призёр чемпионата Европы по самбо, обладатель Кубка мира по самбо, призёр чемпионатов России по дзюдо. Мастер спорта России по дзюдо, заслуженный мастер спорта России по самбо (2018).

В поле Photo загружаем файл:

sajan-hertek.jpg

Поле Sport coach:

Ксения Задворнова — самбо

Поле Info:

Ксения Валерьевна Задворнова (род. 29 декабря 1998, Орск, Оренбургская область[6]) — российская самбистка и дзюдоистка, чемпионка России по дзюдо и самбо 2023 года. Чемпионка мира (2023) и Европы (2024) по самбо. Заслуженный мастер спорта России по самбо. Мастер спорта по дзюдо

В поле Photo загружаем файл:

ksenija-zadvornova.jpg

Поле Sport coach:

Наталья Михайловна Терентьева (Непряева) — лыжи

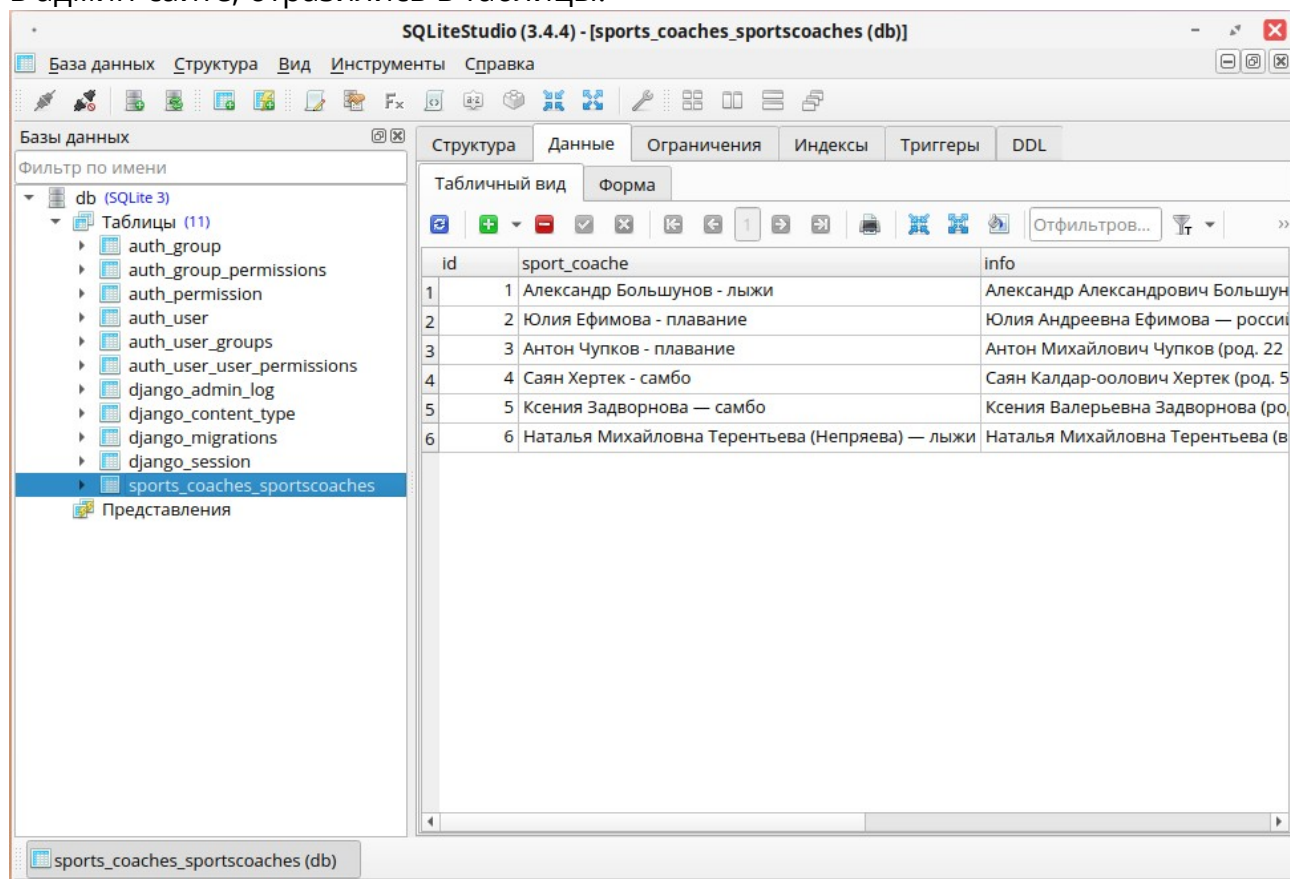
Поле Info:

Наталья Михайловна Терентьева (в девичестве — Непряева, род. 6 сентября 1995, Тверь) — российская лыжница, олимпийская чемпионка в эстафете (2022), трёхкратный призёр Олимпийских игр, трёхкратный призёр чемпионатов мира, обладательница Большого хрустального глобуса (2021/22)[1], первая российская лыжница, выигравшая общий зачёт многодневной гонки «Тур де Ски» (2021/22) [2], двукратная чемпионка мира по бегу на роликовых лыжах (2019, 2021). Заслуженный мастер спорта России (2018)[3]. Универсал, успешно выступает и в дистанционных, и в спринтерских гонках.

В поле Photo загружаем файл:

natalia-neprjaeva.jpg

Ещё рекомендуем скачать программу работы с базой данных SQLiteStudio, перейдя по ссылке <https://sqlitestudio.pl/> и распаковав скаченный архив, перейдём в каталог SQLiteStudio, запустим файл sqlitestudio. После загрузке приложения, нажмём на самую левую кнопку на панели управления, обозначенную как вилка, вставленная в розетку и выберем файл с базой данных db.sqlite3. В иерархической структуре откроем позицию Таблицы и находим таблицу sports_coaches_sportscoaches. Суффикс _sportscoaches был добавлен самим Django. Можно посмотреть, например, структуру указанной таблицы или данные, нажав на соответствующие вкладки. Увидим, что все введенные данные в админ-сайте, отразились в таблице.



Ещё одним способом, с помощью которого можно подправить или дополнить информацию в моделях, это работа интерпретатора Python, встроенная в Django. Запускается этот интерпретатор следующей командой из терминала:

```
python3 manage.py shell
```

Снова перейдём в файл views.py, в котором были созданы обработчики запросов к веб-сайту в виде заглушек и использован класс HttpResponse. С помощью этого класса мы выводили текст с названием веб-страницы. Если мы попробуем выводить полноценный текст веб-страницы со всеми разделами, тегами, таблицами стилей, изображениями, фотографиями, скриптами обработки на

стороне клиента, то поймём, что так делать совершенно глупо. И само предназначение Django предполагает разделение исходного кода страницы на 3 составляющих - модели, представления (контроллеры), шаблоны. Также разделяются так называемые статические файлы. Речь идёт о файлах с каскадными таблицами стилей CSS, файлах с изображениями и файлах со скриптами на JavaScript. С моделями мы уже частично разобрались, создав модель `Sports_coaches` и даже заполнили информацией соответствующей ей таблицу `sports_coaches_sportscoaches` с помощью административного сайта. В контроллерах мы уже не будем использовать класс `HttpResponse`, а воспользуемся обработчиком запросов `render`, который свяжет динамически получаемый код с html шаблоном.

Но сначала напишем ещё одну модель с перечислением видов спорта, к которым принадлежит тот или иной тренер. Понятно, что новая модель должна быть связана с моделью списка тренеров. Поэтому файл `models.py` приобретёт следующий вид:

```
from django.db import models
```

```
class SportsCoaches(models.Model):
```

```
    sport_coach = models.CharField(max_length=250, verbose_name='Спортивный тренер')
```

```
    info = models.TextField(verbose_name='Информация о тренере')
```

```
    photo = models.ImageField(upload_to='media/', verbose_name='Фото тренера')
```

```
    sports_id = models.ForeignKey('Sports', on_delete=models.PROTECT, null=True, verbose_name='Секция')
```

```
    def __str__(self):
```

```
        return self.sport_coach
```

```
class Sports(models.Model):
```

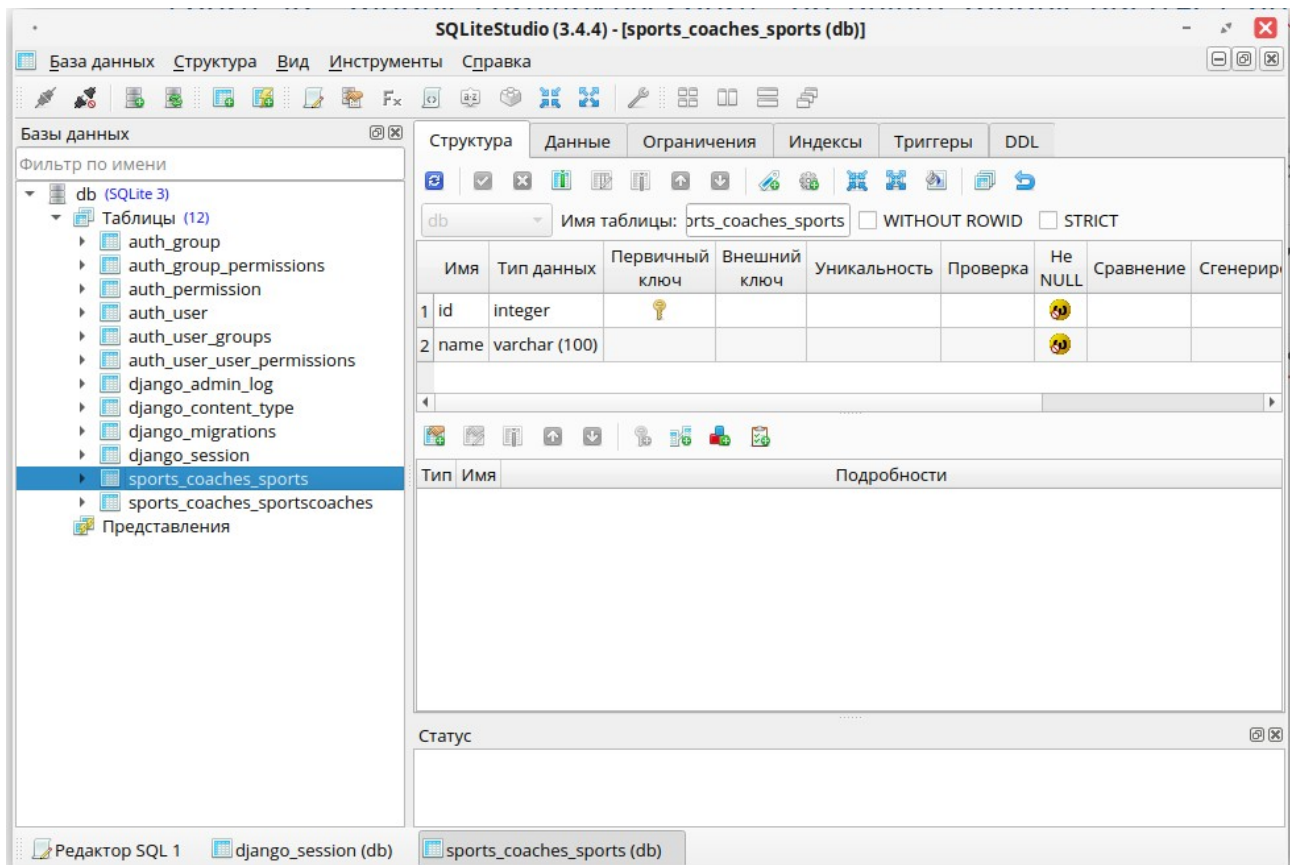
```
    name = models.CharField(max_length=100, db_index=True)
```

```
    def __str__(self):
```

```
        return self.name
```

Связь между моделями `SportsCoaches` и `Sports` осуществляется с помощью дополнительного поля `sports_id` в модели `SportsCoaches`.

Если мы откроем `sqlitestudio`, то увидим, что появилась ещё одна таблица со структурой:



И данными, которые можно записать через админ панель:

	id	name
1	1	Лыжи
2	2	Плавание
3	3	Самбо

Содержание файла views.py расширится и будет иметь вид:

```
from django.shortcuts import render
from .models import SportsCoaches, Sports
from sports_coaches.forms import AddCoachForm
from django.http import HttpResponse
```

```
menu = [{ 'title': "Главная страница", 'url_name': 'home'},
        { 'title': "Информация о нас", 'url_name': 'about'},
        { 'title': "Список тренеров", 'url_name': 'sports_coaches'},
        { 'title': "Добавить тренера", 'url_name': 'add_sport_coach'}
    ]

def index(request):
    context = {
        'menu': menu,
        'title': 'Главная страница',
        'message': 'Добро пожаловать на главную страницу!',
    }
```

```

    }

    return render(request, 'index.html', context)

def about(request):
    context = {
        'menu': menu,
        'title': 'Информация о нас',
    }
    return render(request, 'about.html', context)

def sports_coaches(request):
    queryset = SportsCoaches.objects.all().order_by('sports_id')
    context = {
        'menu': menu,
        'title': 'Список тренеров по различным видам спорта',
        'queryset': queryset,
    }
    return render(request, 'sports_coaches.html', context)

def sport_coach(request, id):
    queryset1 = SportsCoaches.objects.get(pk=id)
    context1 = {
        'menu' : menu,
        'title': 'Информация о тренере',
        'queryset1': queryset1,
    }
    return render(request, 'sport_coach.html', context1)

def add_sport_coach(request):
    if request.method == "POST":
        form = AddCoachForm(request.POST, request.FILES)
        if form.is_valid():
            form.save()
    else:
        form = AddCoachForm()
    context = {
        'menu': menu,
        'title': 'Информация о нас',
        'form': form,
    }
    return render(request, 'add_sport_coach.html', context)

```

В принципе, строчка `from django.http import HttpResponse` нам уже не нужна, если только мы не захотим иметь дополнительные функции и в качестве проверки просто записать для них небольшую заглушку.

Теперь, когда у нас имеются все обработчики запросов, представленных в файле `views.py`, мы можем написать `html` файлы, которые называются шаблонами. В этих шаблонах должны присутствовать специальные теги, вместо которых будет подставляться информация после работы обработчиков из файла `views.py`.

Для написания таких шаблонов на уровне приложения организуется папка с названием `templates`. В ней будут записаны следующие шаблоны:

<code>about.html</code>	- Информация о нас
<code>add_sport_coach.html</code>	- Добавить тренера
<code>index.html</code>	- Главная страница
<code>sport_coach.html</code>	- Информация о тренере
<code>sports_coaches.html</code>	- Список тренеров по различным видам спорта

Когда мы создаём `html` файл, мы прописываем много кода, обязательного для этого формата файла. Чтобы избежать повторов в кодах шаблонов, а повторы нарушают краеугольный закон Django, мы впишем все эти повторы в один файл, и назовём его `base.html`. В дальнейшем мы будем в каждом шаблоне ссылаться на этот файл `base.html`.

Содержание шаблонов будет следующим:

```
base.html
{% load static %}
<html>
  <head>
    <title>{{ title }}</title>
    <link rel="stylesheet" href="{% static 'css/style.css' %}" />

  </head>

  <body>

    <div class="block2">
      <div class="menu">
        <ul>
          <li><a href="{% url 'home' %}"></a></li>
          {% for m in menu %}

            <li><a href="{% url m.url_name %}">{{ m.title }}</a></li>
          {% endfor %}
        </ul>
      </div>
```

```
</div>
<div class="block3">
    <h1 class="">{{ title }}</h1>

    {% block content %}
    {% endblock %}
</div>
</body>
</html>
```

index.html - Главная страница
{% extends 'base.html' %}

```
{% block content %}
    <h3>{{ message }}</h3>
    <div class="slideshow">
<div class="slideshow-item">
    
    <div class="slideshow-item-text">
        <h5>Плавание</h5>
        <p>Юлия Ефимова. Ожидание результатов</p>
    </div>
</div>
<div class="slideshow-item">
    
    <div class="slideshow-item-text">
        <h5>Самбо</h5>
        <p>Удачный бросок</p>
    </div>
</div>
<div class="slideshow-item">
    
    <div class="slideshow-item-text">
        <h5>Лыжи</h5>
        <p>Соревнование женщин</p>
    </div>
</div>
<div class="slideshow-item">
    
    <div class="slideshow-item-text">
        <h5>Самбо</h5>
        <p>Чемпионат мира в Армении 2023 год. Ксения Задворнова в поединке в
весовой категории до 72 кг победила представительницу Белоруссии Анжелу
Жилинскую.</p>
```

</div>
</div>

</div>

<p class="desc">В настоящее время уделяется много внимания развитию массовой физической культуры и спорта.</p>

<p class="desc">Внедрение физической культуры и спорта в целях укрепления здоровья и обеспечения активного отдыха учащихся требует широкого использования имеющихся резервов для привлечения всех школьников и студентов к систематическим физическим занятиям во внеурочное время.</p>

<p class="desc">Нашей общей задачей является привлечение как можно большего количества учащихся к занятиям физической культурой и спортом. Необходимо выработать у них потребность к систематическим занятиям физическими упражнениями.</p>
{% endblock %}

about.html - Информация о нас
{% extends 'base.html' %}

{% block content %}
{{ message }}
<p class="desc">У нас можно записаться в следующие секции:</p>

<h2>Лыжи</h2>

<h4>Сколько стоит экипировка.</h4>

От 5900 ₺: лыжный комбинезон — 3000 ₺, лыжи — 1300 ₺, палки — 400 ₺, ботинки — 1200 ₺.

<p class="desc">Физическая активность на свежем воздухе улучшает работу мозга, насыщая его кислородом.

На занятиях работают все группы мышц, особенно рук, ног и пресса, развивается выносливость и координация тела, тренируется сердечно-сосудистая система. Спортсмены, которые делали упор именно на лыжный бег, как правило, добиваются отличных результатов в биатлоне, полиатлоне и ориентировании.
</p>

<h2>Плавание</h2>

<h4>Сколько стоит экипировка.</h4>

От 628 ₺ для мальчиков и 828 ₺ для девочек. Плавки для мальчиков — 150 ₺, купальник для девочек — 350 ₺, шапочка — 199 ₺, очки — 279 ₺.

<p class="desc">Занятия плаванием под руководством тренера помогут учащимся преодолеть страх воды и отработать правильную технику. Плавание увеличивает объем легких, повышает выносливость, укрепляет верхний плечевой пояс и формирует правильную осанку.

Куда можно перейти. Техника и скорость помогут достичь хороших результатов в водном поло. В триатлоне у пловца есть все шансы оторваться от соперников в воде, но на велогонку и бег сил может не хватить. Можно попробовать себя на открытой воде, но нужно иметь в виду: спортсмену, который привык к идеальным условиям в бассейне, могут помешать волны и сила течения.

</p>

<h2>Самбо</h2>

<h4>Сколько стоит экипировка.</h4>

От 3800 ₺: борцовки — 1300 ₺, куртка и шорты — 2500 ₺.

<p class="desc">Самбо учит самоконтролю, сдержанности, настраивает на мирное решение конфликтов.

Самбо комплексно развивает все группы мышц, положительно влияет на равновесие, координацию, выносливость и дает навыки самообороны. Еще у ребенка формируется стратегическое мышление: все соперники индивидуальны и побеждать во всех поединках по одной отработанной схеме не получится — каждый раз нужно подбирать подход к противнику заново

Самые близкие виды спорта к самбо — боевое самбо и рукопашный бой. Еще можно попробовать скалолазание и гонки с препятствиями: в этих видах спорта огромное значение имеет сила хвата рук, а у самбистов она очень развита.

Подойдет и стритлифтинг: отжимание и подтягивание с гирями — привычные упражнения на тренировках для самбистов. В более взрослом возрасте можно пойти в смешанные единоборства по типу UFC.

</p>

{% endblock %}

sports_coaches.htm - Список тренеров по различным видам спорта

{% extends 'base.html' %}

{% block content %}

<div class="flex-container">

{% for item in queryset %}

<div style="text-align: center;">

{{item.sports_id}}

{% if item.photo %}

<a href="sports_coaches/{{ item.pk }}"

target="_blank"></p>

{% endif %}

<p>{{ item.sport_coach }}</p>

</div>

```
{% endfor %}  
</div>
```

```
{% endblock %}
```

sport_coach.html - Информация о тренере

```
{% extends 'base.html' %}
```

```
{% block content %}
```

```
    <div>  
        <p class="title">{{ queryset1.sport_coach }}</p>  
        <p class="desc">{{ queryset1.info }}</p>  
        {% if queryset1.photo %}  
            <p></p>  
        {% endif %}  
    </div>
```

```
{% endblock %}
```

add_sport_coach.html - Добавить тренера

```
{% extends 'base.html' %}
```

```
{% block content %}
```

```
<div class="desc">
```

```
<form action="{% url 'add_sport_coach' %}" method="post" enctype="multipart/form-  
data">
```

```
    {% csrf_token %}
```

```
    {{ form.as_p }}
```

```
    <button type="submit">Добавить</button>
```

```
</form>
```

```
</div>
```

```
{% endblock %}
```

Всё это хорошо, но никто не отменял связь между url адресом, по которому обращается пользователь и адресом шаблона. Поэтому прописываем следующий код в файле urls.py уровня проекта:

```
from django.contrib import admin
```

```
from django.urls import path
```

```
from sports_coaches.views import index, about, sports_coaches, add_sport_coach,  
sport_coach
```

```
from django.conf import settings
```

```
from django.conf.urls.static import static
```

```
urlpatterns = [
    path ( '', index, name='home'),
    path ( 'sports_coaches/', sports_coaches, name='sports_coaches'),
    path ( 'sports_coaches/sports_coaches/<int:id>/', sport_coach,
name='sport_coach'),
    path ( 'add_sport_coach/', add_sport_coach, name='add_sport_coach'),
    path ( 'about/', about, name= 'about'),
    path ( 'admin/', admin.site.urls),
]
if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL,
document_root=settings.MEDIA_ROOT)
```

Последняя строчка прописана для реализации отображения файлов с фотографиями.

Предварительно нужно создать на уровне проекта каталог media и в нём ещё каталог с таким же именем. Вот туда и нужно поместить фотографии наших тренеров. Почему целесообразно иметь каталог, аналогичный по имени с родительским каталогом. Дело в том, что файлами могут быть не только фотографии, но и, например, файлы с книгами в формате pdf, и резонно их поместить в другой каталог, например, books. В этом случае искать эти файлы по следующему пути media/books/.

Ещё на уровне приложения нужно создать каталог static. В нём должны быть созданы 3 каталога — css (каталог стилей), img (каталог изображений), js (каталог для скриптов JavaScript).

В каталог img помещаем логотип Django django.png, в каталог css помещаем файл стилей style.css.

```
style.css
.form-element {
    font-size: 20;
}
.logo {
    object-fit: cover;
}
.img-max-size {
    width: 300px;
    height: 400px;
    object-fit: cover;
}
.img-min-size {
    width: 75px;
    height: 100px;
    object-fit: cover;
```

```
}
.block2 {
width: 800;
height: 100;
background: #F0E6C2;
margin:auto;
padding:auto;
}

.menu {
    display: flex;
    justify-content: space-between;
}
.menu ul {
    list-style-type: none;
    padding: 0;
    margin: 0;
    display: flex;
    text-align: center;
}
.menu li {
    display: inline-block;
}
.menu ul li:hover, .menu a:hover {
    background: #CBC5C5;
}
.menu li a {
    display: block;
    padding: 8px 15px;
    color: #000;
    text-align: center;
    text-decoration: none;
    font-size: 20px;
}
.block3 {
    width: 800;
background: #FFF;
margin: 0 auto;
padding: 0;
}
.block4 {
    width: 800;
margin: 0 auto;
padding: 0;
}
```

```

.flex-container {
    display: inline-flex;
    flex-direction: row;
    flex-wrap: wrap;
    background-color: #FFF;
}
.flex-container > div {
    background-color: #F0E6C2;
    margin: 10;
    padding: 20;
    font-size: 20;
    width: 110;
    height: 230;
    border-radius: 10px;
}
.desc {
    font-size: 25;
}
.slideshow {
    width: 100%;
    height: 500px;
    position: relative;
    overflow: hidden;
    background: #000;
    margin: 20px 0;
}
.slideshow-item {
    width: 100%;
    height: 100%;
    position: absolute;
    opacity: 0;
    animation: slideanim 40s infinite;
    pointer-events: none;
}
.slideshow-item:nth-child(1),
.slideshow-item:nth-child(1) img {
    animation-delay: 0;
}
.slideshow-item:nth-child(2), .slideshow-item:nth-child(2) img {
    animation-delay: 10s;
}
.slideshow-item:nth-child(3), .slideshow-item:nth-child(3) img {
    animation-delay: 20s;
}

```

```
.slideshow-item:nth-child(4), .slideshow-item:nth-child(4) img {  
    animation-delay: 30s;  
}
```

```
.slideshow-item img {  
    width: 100%;  
    height: 100%;  
    object-fit: cover;  
    animation: zoom 70s infinite;  
}
```

```
.slideshow-item-text {  
    max-width: 50%;  
    position: absolute;  
    bottom: 20px;  
    left: 20px;  
    background-color: rgba(0,0,0,0.7);  
    color: #fff;  
    padding: 20px 30px;  
    font-family: Verdana, sans-serif;  
}
```

```
.slideshow-item-text h5 {  
    font-size: 22px;  
    margin: 0 0 10px 0;  
    color: #BFE2FF;  
}
```

```
.slideshow-item-text p {  
    font-size: 15px;  
    margin-bottom: 10px;  
}
```

```
@keyframes slideanim {  
    12.5%{  
        opacity: 1;  
        pointer-events: auto;  
    }  
    25%{  
        opacity: 1;  
        pointer-events: auto;  
    }  
    37.5%{  
        opacity: 0;  
    }  
}
```

```
@keyframes zoom {
```

```
        50%{
            transform: scale(1.3);
        }
    }
    @media screen and (max-width: 1100px){
        .slideshow-item-text{
            max-width: 75%;
        }
    }
    @media screen and (max-width: 456px){
        .slideshow-item-text {
            bottom: 0;
            left: 0;
            max-width: 100%;
        }
        .slideshow-item-text h5 {
            font-size: 18px;
        }
        .slideshow-item-text p {
            font-size: 13px;
        }
    }
}
```