[1. ВВЕДЕНИЕ 2](#__RefHeading___1)

[2. URLs и Views 3](#__RefHeading___2)

[Создание файла конфигурации (отладка Django в Pycharm) 4](#__RefHeading___3)

[Создание собственного URLconf 4](#__RefHeading___4)

[Динамический URL 4](#__RefHeading___5)

[Конвертеры роутов и редиректы (Converter and Redirect) 6](#__RefHeading___6)

[Функция reverse 6](#__RefHeading___8)

# ВВЕДЕНИЕ

# Установка пакета Django:

pip install django

# Проверка работоспособности Django:

django-admin

После ввода этой команды должно быть выведено:

Type 'django-admin help <subcommand>' for help on a specific subcommand.

Available subcommands:

[django]

check

compilemessages

createcachetable

dbshell

diffsettings

dumpdata

flush

inspectdb

loaddata

makemessages

makemigrations

migrate

runserver

sendtestemail

shell

showmigrations

sqlflush

sqlmigrate

sqlsequencereset

squashmigrations

startapp

startproject

test

testserver

Note that only Django core commands are listed as settings are not properly configured (error: Requested setting INSTALLED\_APPS, but settings are not configured. You must either define the environment variable DJANGO\_SETTINGS\_MODULE or call settings.configure() before accessing settings.).

# Создание проекта (создается папка с названием project\_name, в которой 5 файлов и файл manage.py):

django-admin startproject project\_name

# Про 5 файлов в созданной папке:

manage.py - файл для взаимодействия с проектом через командную строку (execute\_from\_command\_line)

\_\_init\_\_,py - указатель того, что папка является пакетным модулем и ее содержимое можно импортировать

settings.py - файл с настройками проекта

urls.py - файл для регистрации новых страничек на сайте

asgi.py - файл для разворачивания проекта на хосте

wsgi.py - файл для разворачивания проекта на хосте

# Про manage.py:

Команда manage.py - это то же самое, что и команда django-admin, но с дополнительным функционалом

# Запуск локального сервера (из директории в которой расположены папка project\_name и manage.py):

python3 manage.py runserver

После ввода этой команды запускается локальный сервер и прослушивает адрес http://127.0.0.1:8000/ на наличие новых запросов

# Создание приложения (создается новый пакет app\_name в папке проекта в одной плоскости с project\_name и manage.py):

python3 manage.py startapp app\_name

# Про содержимое папки app\_name:

migrations -

models.py -

admin.py - файл для работы с админкой

apps.py - в этом файле лежит название приложения, которое необходимо подключить в разделе INSTALLED\_APPS в файле settings.py (дописать в список)

tests.py - тестирование

views.py - логика приложения

# URLs и Views

# urlpatterns в файле urls.py (URLconf) в папке проекта (самая первая папка, которая создалась после команды manage.py startproject my\_page)

* переменная, представляющая собой список URLов, обрабатывающихся на созданном сайте
* элементы списка выглядят так:

path('admin/', admin.site.urls),

где 'admin/' – URL, admin.site.urls – представление (из файла views.py конкретного приложения)

# views.py

* состоит из функций или классов
* функция принимает обязательный аргумент request (запрос)

def func\_name(responce):

* для отправки ответа функцией нужно импортировать в views.py

from django.http import HttpResponse

и в качестве ответа будет возвращаться экземпляр класса

return HttpResponce('string'),

куда может передаваться html или обычная строка

* необходимо связать каждую функцию с файлом urls.py:

в файле urls.py выполнить

from app\_name import views

и в «представление» в urlpatterns вписать соответствующие views.func\_name (без вызова)

## **Создание файла конфигурации (отладка Django в Pycharm)**

Отладка – остановка программы в указанных точках (красные точки в коде)

Для запуска кода сервера не из командной строки необходимо создать конфигурацию:

Add Configuration – кнопка в правом верхнем углу

«+» в левом верхнем углу и выбрать «Python»

Name – задать название

Working directory – прописать директорию проекта (папка с приложениями и manage.py)

Script path – путь до manage.py (D:\Stepik\Dj\project\my\_page\manage.py)

Parameters – вписать «runserver» для запуска локального сервера, после runserver через пробел можно вписать номер порта, на котором будет запускаться программа (по умолчанию порт 8000)

Запуск дебагера происходит после нажатия на «жучка»

Выполнение кода после назатия на зеленый треугольник снизу слева

Остановка сервера после нажатия на красный квадрат

## Создание собственного URLconf

Для приведения к нормальному виду переменной urlpatterns файла urls.py проекта необходимо использовать include(), как прописано в самом файле urls.py:

* создать в папке приложения файл urls.py, в котором будут содержаться все странички, относящиеся к одному домену (в нашем случае horoscope/): [path('aries/', views.aries), path('taurus/', views.taurus), ...], при этом сделать в этом файле from . import views (файлы urls.py и views.py расположены в одной и той же папке приложения)
* вместо большого количества элементов списка, относящихся к одному и тому же доменному адресу необходимо прописать: path('horoscope/', include('horoscope.urls'))
* в include прописывается путь к роутам конкретного приложения, в нашем случае это horoscope.urls (связь с файлом urls.py приложения horoscope)

## Динамический URL

# в файле urls.py приложения заменить содержимое переменной urlpatterns на:

urlpatterns = [path('<zodiac\_sign>/', views.get\_info\_by\_zodiac\_sign)]

# в urlpatterns вместо захардкоженных url ('aries/', taurus/', ...) теперь передается параметр:

'<zodiac\_sign>/'

# в файле views.py приложения создать новое представление в замен множества старых:

def get\_info\_by\_zodiac\_sign(request, zodiac\_sign):

description = zodiac\_dict.get(zodiac\_sign)

if description:

return HttpResponse(description)

else:

return HttpResponseNotFound(f'{zodiac\_sign} - unknown zodiac sign')

# параметр zodiac\_sign является обязательным параметром для представления

# для отправки ответа «404 NOT FOUND» функцией нужно импортировать в views.py

from django.http import HttpResponseNotFound

и в качестве ответа будет возвращаться экземпляр класса

return HttpResponseNotFound(f'{zodiac\_sign} - unknown zodiac sign')

# для удобства использования возможные URLы и ответы на них записали в словарь

zodiac\_dict = {

'aries': 'Овен - первый знак зодиака, планета Марс (с 21 марта по 20 апреля).',  
'taurus': 'Телец - второй знак зодиака, планета Венера (с 21 апреля по 21 мая).',  
'gemini': 'Близнецы - третий знак зодиака, планета Меркурий (с 22 мая по 21 июня).',  
'cancer': 'Рак - четвёртый знак зодиака, Луна (с 22 июня по 22 июля).',  
'leo': 'Лев - пятый знак зодиака, солнце (с 23 июля по 21 августа).',  
'virgo': 'Дева - шестой знак зодиака, планета Меркурий (с 22 августа по 23 сентября).',  
'libra': ' Весы - седьмой знак зодиака, планета Венера (с 24 сентября по 23 октября).',  
'scorpio': 'Скорпион - восьмой знак зодиака, планета Марс (с 24 октября по 22 ноября).',  
'sagittarius': 'Стрелец - девятый знак зодиака, планета Юпитер (с 23 ноября по 22 декабря).',  
'capricorn': 'Козерог - десятый знак зодиака, планета Сатурн (с 23 декабря по 20 января).',  
'aquarius': 'Водолей - одиннадцатый знак зодиака, планеты Уран и Сатурн (с 21 января по 19 февраля).',

'pisces': 'Рыбы - двенадцатый знак зодиака, планеты Юпитер (с 20 февраля по 20 марта).'

}

## Конвертеры роутов и редиректы ([Converter and Redirect](https://github.com/annasidorova97/Django_stepik/commit/6d0f079db7dc1b0d32924fb5f3b5e8b05b367969))

# в файле urls.py приложения можно применить преобразование вводимого роута (первым выполняется верхнее преобразование, а ежу при неуспешном исходи - следующее):

urlpatterns = [

path('<**int:**zodiac\_sign>', views.get\_info\_by\_number\_zodiac\_sign),

path('<**str:**zodiac\_sign>',views.get\_info\_by\_zodiac\_sign)

]

# при этом можно аннотировать тип вводимого роута в представлении:

def get\_info\_by\_zodiac\_sign(request, zodiac\_sign**: str**):

description = zodiac\_dict.get(zodiac\_sign)

if description:

return HttpResponse(description)

else:

return HttpResponseNotFound(f'{zodiac\_sign} - unknown zodiac sign')

def get\_info\_by\_number\_zodiac\_sign(request, zodiac\_sign**: int**):

if zodiac\_sign > len(list(zodiac\_dict)):

return HttpResponseNotFound(f'{zodiac\_sign} - false number of zodiac sign')

zodiac\_name = list(zodiac\_dict)[zodiac\_sign – 1]

return HttpResponseRedirect(f'/horoscope/{zodiac\_name}')

# для применения редиректа неодходимо импортировать:

from django.http import HttpResponseRedirect

# в нашем случае при задании знака гороскопа цифрой в строке ввода браузера происходит перенаправление на адрес (тут можно указать любой существующий URL):

return HttpResponseRedirect(f'/horoscope/{zodiac\_name}')

## Функция reverse

# для того, чтобы не хардкодить при редиректах (жестко прописывается часть URLа):

return HttpResponseRedirect(f'**/horoscope/**{zodiac\_name}')

# необходимо дать название роуту, к которому мы будем в дальнейшем обращаться при редиректах, в файле urls.py приложения:

path('<str:zodiac\_sign>', views.get\_info\_by\_zodiac\_sign, **name='horoscope\_name'**)

# далее в файле представления views.py приложения импортировать reverse:

from django.urls import reverse

# теперь к доменной части URLа можно обращаться по имени, при этом любые её изменения не повлияют на остальной код (args должна быть упорядоченной коллекцией):

def get\_info\_by\_number\_zodiac\_sign(request, zodiac\_sign: int):

if zodiac\_sign > len(list(zodiac\_dict)):

return HttpResponseNotFound(f'{zodiac\_sign} - false number of zodiac sign')

zodiac\_name = list(zodiac\_dict)[zodiac\_sign – 1]

**redirect\_url = reverse('horoscope\_name', args=[zodiac\_name])**

**return HttpResponseRedirect(redirect\_url)**