

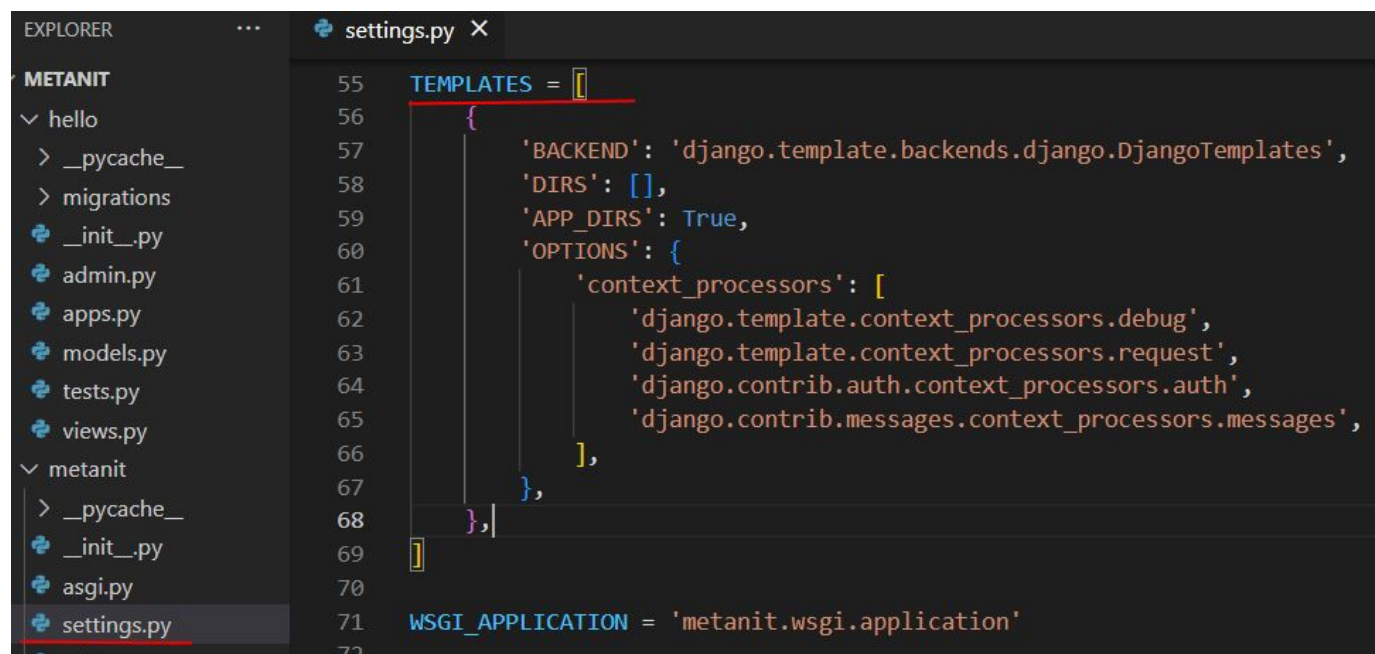
# Шаблоны

## Создание и использование шаблонов

Шаблоны (templates) отвечают за формирование внешнего вида приложения. Они предоставляют специальный синтаксис, который позволяет внедрять данные в код HTML.

Допустим, у нас есть проект metanit, и в нем определено одно приложение - hello

Настройка функциональности шаблонов в проекте Django производится в файле settings.py. с помощью переменной TEMPLATES. Так, по умолчанию переменная TEMPLATES в файле settings.py имеет следующее определение:



```
55 TEMPLATES = [
56     {
57         'BACKEND': 'django.template.backends.django.DjangoTemplates',
58         'DIRS': [],
59         'APP_DIRS': True,
60         'OPTIONS': {
61             'context_processors': [
62                 'django.template.context_processors.debug',
63                 'django.template.context_processors.request',
64                 'django.contrib.auth.context_processors.auth',
65                 'django.contrib.messages.context_processors.messages',
66             ],
67         },
68     ],
69 ]
70
71 WSGI_APPLICATION = 'metanit.wsgi.application'
72
```

Данная переменная принимает список конфигураций для каждого движка шаблонов. По умолчанию определена одна конфигурация, которая имеет следующие параметры

**BACKEND:** движок шаблонов. По умолчанию применяется встроенный движок `django.template.backends.django.DjangoTemplates`

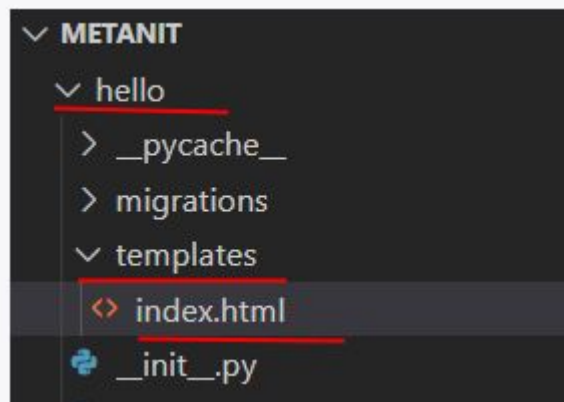
**DIRS:** определяет список каталогов, где движок шаблонов будет искать файлы шаблонов. По умолчанию пустой список

**APP\_DIRS:** указывает, будет ли движок шаблонов искать шаблоны внутри папок приложений в папке `templates`.

**OPTIONS:** определяет дополнительный список параметров

Итак, в конфигурации по умолчанию параметр `APP_DIRS` имеет значение `True`, а это значит, что движок шаблонов будет также искать нужные файлы шаблонов в папке приложения в каталоге `templates`. То есть по умолчанию мы уже имеем настроенную конфигурацию, готовую к использованию шаблонов. Теперь определим сами шаблоны.

Добавим в папку приложения каталог `templates`. А в нем определим файл `index.html`:



Далее в файле index.html определим следующий код:

```
<!DOCTYPE html>
<html>
<head>
  <title>Django на METANIT.COM</title>
  <meta charset="utf-8" />
</head>
<body>
  <h2>Hello METANIT.COM</h2>
</body>
</html>
```

По сути это обычная веб-страница, которая содержит код html. Теперь используем эту страницу для отправки ответа пользователю. И для этого перейдем в приложении hello к файлу views.py, который определяет функции для обработки запроса. Изменим этот файл следующим образом:

```
from django.shortcuts import render

def index(request):
    return render(request, "index.html")
```

Из модуля django.shortcuts импортируется функция render.

Функция index вызывает функцию render, которой передаются объект запроса request и путь к файлу шаблона в рамках папки templates - "index.html"

В файле urls.py проекта пропишем сопоставление функции index с запросом к корню веб-приложения:

```
from django.urls import path
from firstapp import views

urlpatterns = [
    path("", views.index),
]
```

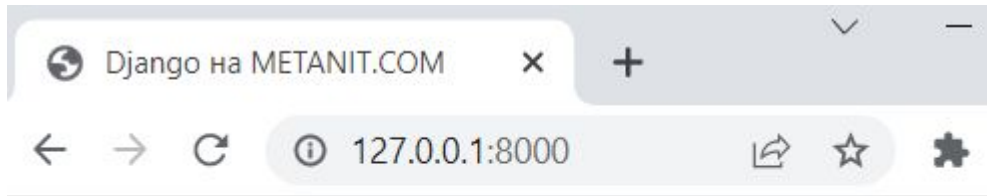
The screenshot shows a code editor with three files open: `views.py`, `index.html`, and `urls.py`. The `views.py` file contains a function `index(request)` that returns `render(request, "index.html")`. The `index.html` file contains an HTML template with a title "Django на METANIT.COM" and a body containing `<h1>Hello METANIT.COM</h1>`. The `urls.py` file contains a `urlpatterns` list with a `path("", views.index)` entry. A red arrow points from the `return render(request, "index.html")` line in `views.py` to the `index.html` file in the Explorer pane.

```
EXPLORER
METANIT
  hello
    __pycache__
    migrations
    templates
      index.html
    __init__.py
    admin.py
    apps.py
    models.py
    tests.py
    views.py
  metanit
    __pycache__
    __init__.py
    asgi.py
    settings.py

views.py
1 from django.shortcuts import render
2
3 def index(request):
4     return render(request, "index.html")

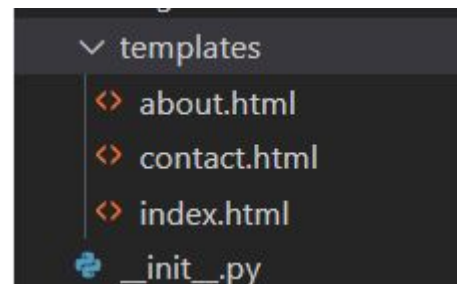
index.html
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <title>Django на METANIT.COM</title>
5     <meta charset="utf-8" />
6 </head>
7 <body>
8     <h1>Hello METANIT.COM</h1>
9 </body>
10 </html>

urls.py
1 from django.urls import path
2 from hello import views
3
4 urlpatterns = [
5     path("", views.index),
6 ]
```



**Hello METANIT.COM**

Подобным образом можно указать и другие шаблоны. Например, в папку templates добавим еще две страницы: about.html и contact.html



И также в файле views.py определим функции, которые используют данные шаблоны:

```
from django.shortcuts import render

def index(request):
    return render(request, "index.html")

def about(request):
    return render(request, "about.html")

def contact(request):
    return render(request, "contact.html")
```

А в файле urls.py свяжем функции с маршрутами:

```
from django.urls import path
from hello import views

urlpatterns = [
    path("", views.index),
    path("about/", views.about),
    path("contact/", views.contact),
]
```

## TemplateResponse

Выше для генерации шаблона применялась функция render(), которая является наиболее распространенным вариантом. Однако также мы можем использовать класс TemplateResponse:

```
from django.template.response import TemplateResponse

def index(request):
    return TemplateResponse(request, "index.html")
```

## Передача данных в шаблоны

Одним из преимуществ шаблонов является то, что можно передать в них динамически из представлений различные данные. Для вывода данных в шаблоне могут использоваться различные способы. Для вывода самых простых данных применяется двойная фигурная скобка: `{{ название_объекта }}`

Например, пусть в проекте есть папка `templates`, в которой содержится шаблон `index.html`:

Определим в файле `index.html` следующий код:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Django на METANIT.COM</title>
</head>
<body>
  <h2>{{ header }}</h2>
  <p>{{ message }}</p>
</body>
</html>
```

Здесь используется две переменных: `message` и `header`. Они будут передаваться из представления.



Чтобы из функции-представления передать данные в шаблон применяется третий параметр функции **render**, который еще называется **context** и который представляет словарь. Например, изменим файл **views.py** следующим образом:

```
from django.shortcuts import render

def index(request):
    data = {"header": "Hello Django", "message": "Welcome to Python"}
    return render(request, "index.html", context=data)
```

В шаблоне используются две переменные, соответственно словарь, который передается в функцию `render` через параметр `context`, теперь содержит два значения с ключами `header` и `message`.

В результате при обращении к корню веб-приложения вывод в браузере будет:



# Hello Django

Welcome to Python



## Передача сложных данных

Рассмотрим передачу более сложных данных. Допустим, в представлении передаются следующие данн

```
from django.shortcuts import render

def index(request):
    header = "Данные пользователя"           # обычная переменная
    langs = ["Python", "Java", "C#"]        # список
    user = {"name": "Tom", "age": 23}       # словарь
    address = ("Абрикосовая", 23, 45)      # кортеж

    data = {"header": header, "langs": langs, "user": user, "address": address}
    return render(request, "index.html", context=data)
```

В качестве третьего параметра в функцию `render` нам надо передать словарь, поэтому все данные оборачиваются в словарь и в таком виде передаются в шаблон.

В этом случае шаблон может выглядеть, следующим образом:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Django на METANIT.COM</title>
</head>
<body>
  <h1>{{ header }}</h1>
  <p>Имя: {{ user.name}} Возраст: {{user.age}}</p>
  <p>Адресс: ул. {{address.0}}, д. {{address.1}}, кв. {{address.2}}</p>
  <p>Языки: {{langs.0}}, {{langs.1}}</p>
</body>
</html>
```

## TemplateResponse

Если для генерации шаблона применяется класс `TemplateResponse`, то в его конструктор также через третий параметр можно передать данные для шаблона:

```
from django.template.response import TemplateResponse

def index(request):
    header = "Данные пользователя"           # обычная переменная
    langs = ["Python", "Java", "C#"]        # список
    user = {"name": "Tom", "age": 23}       # словарь
    address = ("Абрикосовая", 23, 45)      # кортеж

    data = {"header": header, "langs": langs, "user": user, "address": address}
    return TemplateResponse(request, "index.html", data)
```

## Передача объектов классов

Подобным образом можно передавать в шаблоны объекты своих классов. Например, определение функции-представления:

```
from django.shortcuts import render

def index(request):
    return render(request, "index.html", context = {"person": Person("Tom")})

class Person:

    def __init__(self, name):
        self.name = name # ИМЯ ЧЕЛОВЕКА
```

В шаблоне index.html мы можем обращаться к функциональности объекта, например, к его атрибуту name:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Django на METANIT.COM</title>
</head>
<body>
  <h1>Person {{ person.name }}</h1>
</body>
</html>
```

Здесь определяется класс Person, в конструкторе которого передается некоторое значение для атрибута name. В функции index в шаблон передается объект с ключом "person".



## Встроенные теги шаблонов

Django предоставляет возможность использовать в шаблонах ряд специальных тегов, которые упрощают вывод некоторых данных. Рассмотрим некоторые наиболее используемые теги.

### autoescape

Тег autoescape позволяет автоматически экранировать ряд символов HTML и тем самым сделать вывод на страницу более безопасным. В частности, производятся следующие замены:

< заменяется на &lt;

> заменяется на &gt;

' (одинарная кавычка) заменяется на &#x27;

" (двойная кавычка) заменяется на &quot;

& заменяется на &amp;

Например, пусть у нас будет определен следующий шаблон index.html:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Django на METANIT.COM</title>
</head>
<body>
  {{ body }}

  {% autoescape off %}
  {{ body }}
  {% endautoescape %}
</body>
</html>
```

### функция- представление

```
from django.shortcuts import render

def index(request):
    return render(request, "index.html", context = {"body": "<h1>Hello World!</h1>"})
```



<h1>Hello World!</h1>

**Hello World!**

## Комментарии

Для определения комментариев в шаблоне применяется тег `comment`: все, что помещается между тегами `{% comment %}` и `{% endcomment %}`, игнорируется при генерации html-страницы.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Django на METANIT.COM</title>
</head>
<body>

  {% comment %}
    <p>Вывод содержимого в шаблон Django</p>
  {% endcomment %}

  {{ body }}
</body>
</html>
```

## extends

Обозначает, что этот шаблон расширяет родительский шаблон.

Этот тег можно использовать двумя способами:

`{% extends "base.html" %}` (в кавычках) использует буквальное значение "base.html" в качестве имени родительского шаблона для расширения.

`{% extends variable %}` использует значение `variable`. Если переменная вычисляется как строка, Django будет использовать эту строку как имя родительского шаблона. Если переменная оценивается как объект `Template`, Django будет использовать этот объект в качестве родительского шаблона.

```
dir1/
  template.html
  base2.html
  my/
    base3.html
base1.html
```

```
{% extends "./base2.html" %}
{% extends "../base1.html" %}
{% extends "./my/base3.html" %}
```

## block

Определяет блок, который может быть переопределен дочерними шаблонами.

Например создадим шаблон, который мы назовем base.html, он определяет скелет HTML-документа, который можно использовать для страницы с двумя столбцами. Задача «дочерних» шаблонов - заполнить пустые блоки содержимым.

В этом примере тег block определяет три блока, которые могут быть заполнены дочерними шаблонами. Все, что делает тег block, - это сообщает механизму шаблонов, что дочерний шаблон может переопределить эти части шаблона.

## Дочерний шаблон

```
<!DOCTYPE html>
<html lang="en">
<head>
  <link rel="stylesheet" href="style.css">
  <title>{% block title %}My amazing site{% endblock %}</title>
</head>

<body>
  <div id="sidebar">
    {% block sidebar %}
    <ul>
      <li><a href="/">Home</a></li>
      <li><a href="/blog/">Blog</a></li>
    </ul>
    {% endblock %}
  </div>

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

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

{% block title %}My amazing blog{% endblock %}

{% block content %}
{% for entry in blog_entries %}
  <h2>{{ entry.title }}</h2>
  <p>{{ entry.body }}</p>
{% endfor %}
{% endblock %}
```



## filter

Фильтрует содержимое блока с помощью одного или нескольких фильтров. С помощью конвейеров можно указать несколько фильтров, а фильтры могут иметь аргументы, как и в синтаксисе переменных.

Обратите внимание, что блок включает весь текст между тегами filter и endfilter.

```
{% filter force_escape|lower %}  
    This text will be HTML-escaped, and will appear in all lowercase.  
{% endfilter %}
```

## include

Загружает шаблон и отображает его с текущим контекстом. Это способ «включения» других шаблонов в шаблон.

Имя шаблона может быть переменной или жестко закодированной (заключенной в кавычки) строкой в одинарных или двойных кавычках.

```
{% include "foo/bar.html" %}
```

```
{% include template_name %}
```

можно передать дополнительный контекст в шаблон, используя ключевое слово arguments

```
{% include "name_snippet.html" with person="Jane" greeting="Hello" %}
```

Шаблон name\_snippet.html

```
{{ greeting }}, {{ person|default:"friend" }}!
```



## url

Возвращает ссылку на абсолютный путь (URL без имени домена), соответствующую заданному представлению и необязательным параметрам. Любые специальные символы в полученном пути будут закодированы с использованием

```
{% url 'some-url-name' v1 v2 %}
```

Например, предположим, что у вас есть представление `app_views.client`, `URLconf` которого принимает идентификатор клиента (здесь `client()` - это метод внутри файла представлений `app_views.py`). Строка `URLconf` может выглядеть

```
path('client/<int:id>/', app_views.client, name='app-views-client')
```

Если `URLconf` этого приложения включен в `URLconf` проекта по следующему пути:

```
path('clients/', include('project_name.app_name.urls'))
```

затем в шаблоне можно создать ссылку на это представление следующим образом:

```
{% url 'app-views-client' client.id %}
```

Тег шаблона выведет строку `/clients/client/123/`.

синтаксис `{% url ... as var %}` не вызовет ошибку, если представление отсутствует.

```
{% url 'some-url-name' as the_url %}
{% if the_url %}
  <a href="{{ the_url }}">Link to optional stuff</a>
{% endif %}
```

## dictsort

Принимает список словарей и возвращает этот список, отсортированный по ключу, указанному в аргументе.

```
{{ value|dictsort:"name" }}
```

Например:

value пришло:

```
[
  {'name': 'zed', 'age': 19},
  {'name': 'amy', 'age': 22},
  {'name': 'joe', 'age': 31},
]
```

результат:

```
[
  {'name': 'amy', 'age': 22},
  {'name': 'joe', 'age': 31},
  {'name': 'zed', 'age': 19},
]
```

## join

Объединяет список со строкой, как в Python `str.join(list)`

Например:

```
{{ value|join:" // " }}
```

Если value является списком ['a', 'b', 'c'], то выходом будет строка "a // b // c".

## if..else

Тег if оформляется в виде блока:

```
{% if условие %}  
    содержимое блока  
{% endif %}
```

Этот тег оценивает некоторое условие, которое должно возвращать True или False: если возвращается True, то выводится содержимое блока `{% if %}`.

Например, пусть в представлении передаются в шаблон некоторые значения:

```
from django.shortcuts import render  
  
def index(request):  
    data = {"n" : 5}  
    return render(request, "index.html", context=data)
```

В шаблоне в зависимости от значения переменной `n` мы можем выводить определенный контент:

```
{% if n > 0 %}  
    <p>Число положительное</p>  
{% endif %}
```

```
{% if n > 0 %}  
    <p>Число положительное</p>  
{% else %}  
    <p>Число отрицательное или равно нулю</p>  
{% endif %}
```

```
{% if n > 0 %}  
    <p>Число положительное</p>  
{% elif n < 0 %}  
    <p>Число отрицательное</p>  
{% else %}  
    <p>Число равно нулю</p>  
{% endif %}
```

Стоит отметить, что в условии, которое передается тегу `if`, можно применять ряд операторов, которые возвращают True или False: `==`, `!=`, `<`, `>`, `<=`, `>=`, `in`, `not in`, `is`, `is not`, `and`, `or`, `not`.

## Циклы

Тег for позволяет создавать цикл

```
{% for element in collection %}  
    действия с element  
{% endfor %}
```

Например, пусть из представления в шаблон передается некоторый список:

```
from django.shortcuts import render  
  
def index(request):  
    langs = ["Python", "JavaScript", "Java", "C#", "C++"]  
    return render(request, "index.html", context={"langs": langs})
```

Выведем элементы списка langs в шаблоне с помощью тега **for**:

```
<!DOCTYPE html>  
<html>  
<head>  
    <meta charset="utf-8" />  
    <title>Django on METANIT.COM</title>  
</head>  
<body>  
    <ul>  
        {% for lan in langs %}  
        <li>{{ lan }}</li>  
        {% endfor %}  
    </ul>  
</body>  
</html>
```



- Python
- JavaScript
- Java
- C#
- C++

Когда переданная из представления в шаблон коллекция окажется пустой можно использовать тег `{% empty %}`:

```
<ul>
  {% for lan in langs %}
  <li>{{ lan }}</li>
  {% empty %}
  <li>Langs array is empty</li>
  {% endfor %}
</ul>
```

Подобным образом можно перебирать в шаблоне другие коллекции, например, словари:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Django on METANIT.COM</title>
</head>
<body>
  {% for key, value in data.items %}
  <p>{{ key }}: {{ value }}</p>
  {% endfor %}
</body>
</html>
```

В данном случае перебирается словарь `data`, который передается из представления:

```
from django.shortcuts import render

def index(request):
    data = {"red": "красный", "green": "зеленый", "blue": "синий"}
    return render(request, "index.html", context={"data": data})
```

## Определение переменных

Тег `{% with %}` позволяет определить переменную и использовать ее внутри блока тега. Например:

```
{% with name="Tom" age=29 %}  
  <div>  
    <p>Name: {{ name }}</p>  
    <p>Age: {{ age }}</p>  
  </div>  
{% endwith %}
```

В данном случае определены две переменных: `name` и `age`, которые можно использовать внутри блока `{% with%} ... {% endwith %}`. Однако вне этого блока эти переменные использоваться не могут.

## Даты

Тег `{% now "формат_данных" %}` позволяет вывести системное время. В качестве параметра тегу `now` передается формат данных, который указывает, как форматировать время и дату.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Django on METANIT.COM</title>
</head>
<body>
  <p>{% now "Y" %}</p>
  <p>{% now "F j Y" %}</p>
  <p>{% now "N, j, Y" %}</p>
  <p>{% now "N j, Y, P" %}</p>
</body>
</html>
```



2022

August 17 2022

Aug., 17, 2022

Aug. 17, 2022, 6:01 p.m.

Символ "Y" передает год в виде четырех цифр, "y" - берет из года последние две цифры. "F" - полное название месяца, "N" - сокращенное название месяца. "j" - день месяца. "P" - время. Все возможные форматы для вывода даты и времени можно посмотреть в документации -

<https://django.fun/ru/docs/django/4.1/>

В качестве времени Django использует переменную `TIME_ZONE` из файла `settings.py`, которую имеет следующее определение:

```
TIME_ZONE = 'UTC'
```

для московского часового пояса

```
TIME_ZONE = 'Europe/Moscow'
```

<https://django.fun/ru/docs/django/4.1/ref/templates/builtins/>



## Задание

Создать Django-приложение:

- / - открывает шаблон index.html, где находится информация о вас на данный момент: где учитесь, группа, специальность, кем хотите быть после выпуска
- about – передать в шаблон about.html данные о себе: ФИО, рост, вес, возраст
- contacts – передать в шаблон contacts.html словарь со своими контактными данными, где ключ – это способ связи (телефон, соцсеть и т.д), значение – данные, по которым можно связаться (номер, адрес, ссылка)
- создать шаблон form.html с формой регистрации (можно пока без стилей). Обязательные поля: ФИО, email, пол (радио кнопки), кнопка отправки (пока не нужно организовывать алгоритм отправки, делаем внешний вид)
- подключить форму на главную страницу с помощью встроенных тегов.
- добавить функционал вывода данных на странице contacts с помощью циклов.