Обработка запросов и шаблоны

План занятия

1. Работа с конфигом Django
2. Параметры запроса
3. Введение в шаблоны
4. Пагинация

Работа с конфигом Django

Как работать с настройками в Django

В settings.py можно добавлять **свои собственные переменные** и потом пользоваться ими в любом удобном месте.

Чтобы получить значения из конфигурации, необходимо обращаться к **полям в объекте** settings:

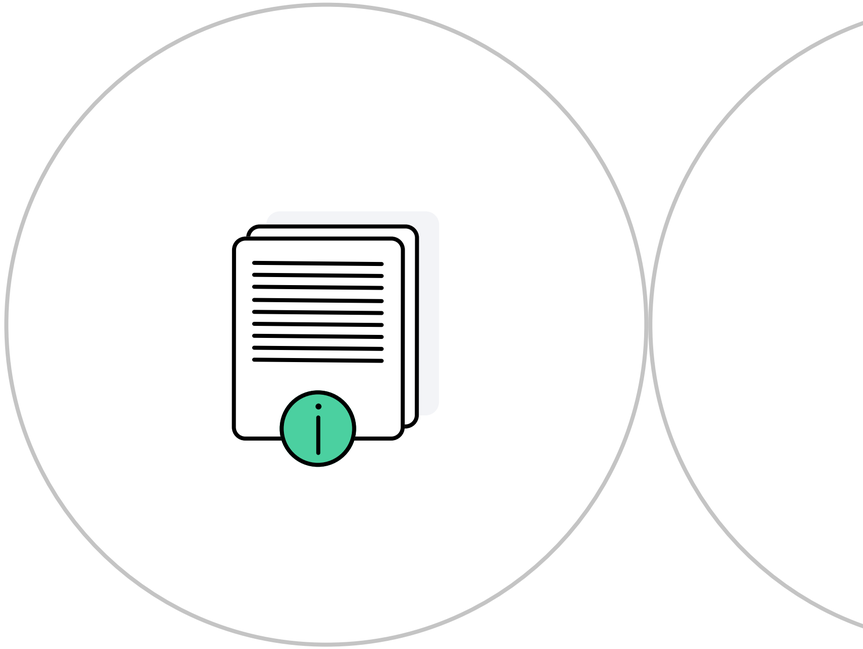
```
# именно так надо импортировать настройки
from django.conf import settings
from django.http import HttpResponse

def hello_view(request):
    msg = f'Свяжитесь с админом {settings.CONTANCT_EMAIL}'
    return HttpResponse('Всем привет! Я Django! ' + msg)
```

Параметры запроса

Однотипные страницы

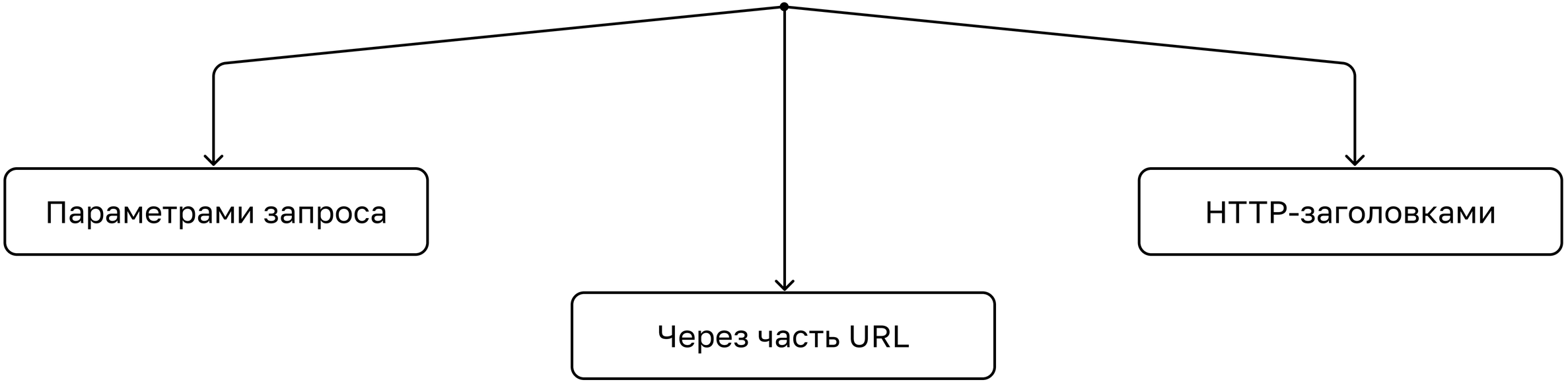
Если на сайте представлено много **однотипных страниц**, например, карточка товара или профили пользователей, то эти однотипные страницы могут обрабатываться **единственным** view.

В таком случае view будет извлекать динамические параметры из запроса и возвращать пользователю ответ



Добавляем динамики

Варианты добавления динамики в запросы



Параметры запроса

Передаются **после** символа ? в конце запроса:

```
https://example.org?name=ivan
```

В Django их можно получить из **объекта** request:

```
request.GET.get('name')
```

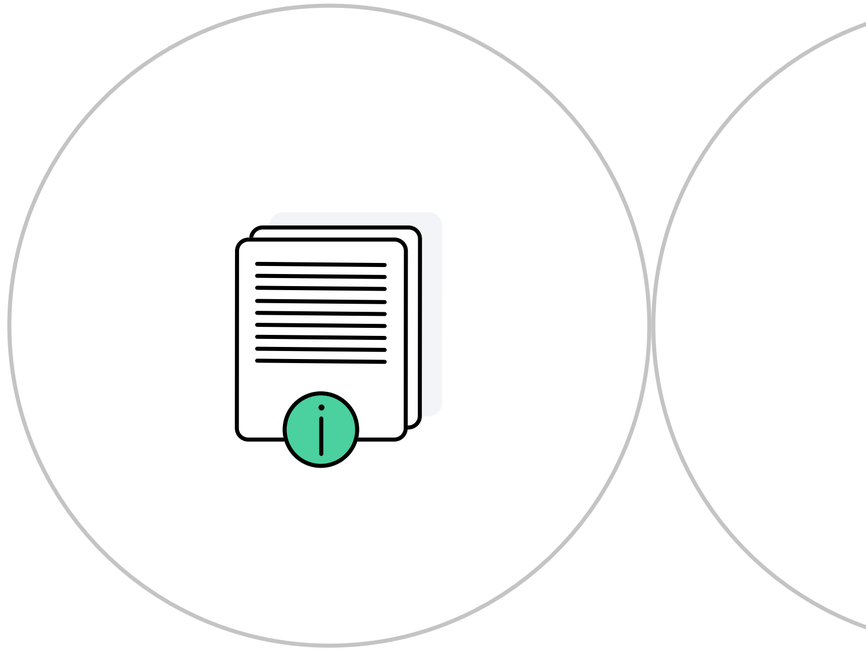
Дополнительная информация про request:
https://docs.djangoproject.com/en/3.2/ref/request-response/#django.http.HttpRequest.GET

Обработка параметров запроса

```
def home_view(request):
name = request.GET.get('name')
if name:
    response = f'Здравствуйте, {name}!'
else:
    response = f'Пожалуйста, представьтесь'
return HttpResponse(response)
```

Динамический URL

Часть URL также может быть динамической.
Django **умеет** парсить* URL в соответствии с заданным шаблоном и передавать полученные параметры во view.
Пример: https://docs.djangoproject.com/en/3.2/topics/http/urls/#example



*Парсить — собирать и систематизировать размещённую на определённых сайтах информацию с помощью специальных программ, автоматизирующих процесс

Динамический URL: пример

```
from django.urls import path

from . import views

urlpatterns = [
    path('hello/<name>/', views.hello),
    path('articles/<int:year>/<int:month>/<int:day>/', views.article_detail),
    path('articles/<int:year>-<int:month>-<int:day>/', views.article_detail),
]

# views.py
def hello(request, name):
response = f'Здравствуйте, {name}!'
return HttpResponse(response)

def article_detail(request, year, month, day):
response = f'Статья за {year} год {month} месяц {day} день!'
return HttpResponse(response)
```

Введение в шаблоны

Как создать HTML

HTML генерируется в результате **рендеринга специальных шаблонов**.
Конфигурация шаблонов задаётся через опцию TEMPLATES в конфиге:

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                ...
            ],
        },
    },
]
```

Как создать HTML

Важно: шаблоны внутри одного приложения создаются в отдельном пространстве имён.

Если приложение называется base, то:

```
templates
|-- base
    |-- home.html
```

Дополнительная информация про Templates:
https://docs.djangoproject.com/en/3.2/topics/templates/

Шаблоны

Шаблоны — **это расширенный HTML.**
В шаблоне можно выполнить подстановку значения переменной:
{{ название_переменной }}

Через точку **можно получить** атрибут, ключ или вызвать метод:

```
{{ user.first_name }}  # атрибут
{{ name.strip }}  # метод (без скобок и без параметров!)
{{ some_dict.key }}  # ключ
```

Методы вызываются **без параметров.**
Пример текста:

```
My first name is {{ user.first_name }}
```

Функция render

render — функция, позволяющая **отрендерить шаблон** с определённым контекстом.

Контекст шаблона — это параметры, которые будут подставлены в HTML динамически.

Даже если вы явно не передаёте никакие параметры, определённые глобальные объекты будут заданы через context_processors

Дополнительная информация про render:
https://docs.djangoproject.com/en/3.2/topics/http/shortcuts/#render

Использование функции render

```
def home(request):
    context = {
        'first_name': 'Jake',
        'last_name': 'The Dog'
    }
    return render(request, 'home.html', context)
```

Результат

My first name is Jake. My last name is The Dog

Шаблоны. Теги

Теги в шаблонах — это **расширение** для динамической генерации HTML-страниц.

Теги похожи на Python-код внутри HTML-файла:

```
{% for obj in some_list %}
  <tr class="table-row">{% include "subtemplate.html" %}</tr>
{% endfor %}
```

Теги указываются в {% ... %}

Полный список тегов можно найти в документации:
https://docs.djangoproject.com/en/3.2/ref/templates/builtins/

Примеры тегов

for позволяет **итерироваться** по объекту:

```
<ul>
{% for student in student_list %}
    <li>{{ student.rating }}</li>
{% endfor %}
</ul>
```

if позволяет **проверять** условие:

```
{% if user.is_authenticated %}
    <p>Здравствуйте, {{ user.username }}!</p>
{% endif %}
```

Больше возможностей

Какие ещё есть возможности у шаблонов в Django:

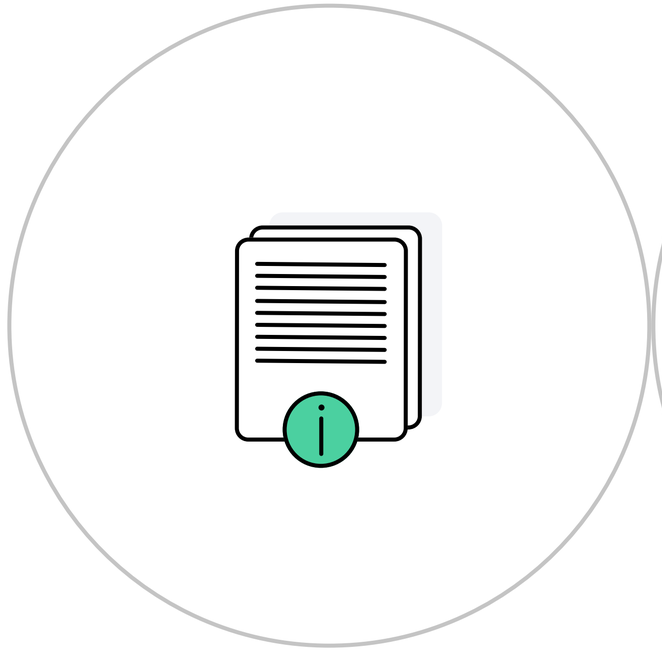
- фильтры:
  https://docs.djangoproject.com/en/3.1/ref/templates/builtins/#built-in-filter-reference
- наследование:
  https://docs.djangoproject.com/en/3.2/ref/templates/builtins/#extends
- композиция:
  https://docs.djangoproject.com/en/3.2/ref/templates/builtins/#include
- собственные теги и фильтры:
  https://docs.djangoproject.com/en/3.1/howto/custom-template-tags/

Пагинация

Пагинация

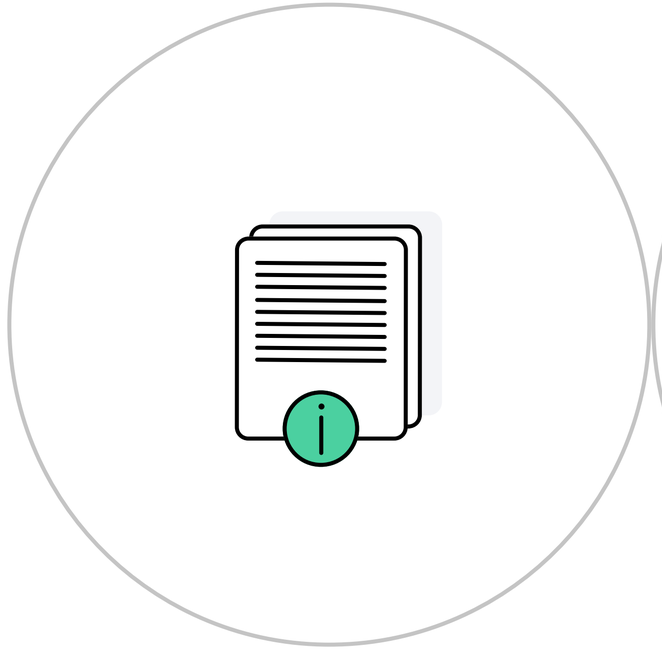
Пагинация — это способ выводить контент **постранично**

Пример записи в блоге. На каждой странице содержится определённое число записей, навигация может осуществляться как на конкретную страницу, так и просто на следующую или предыдущую



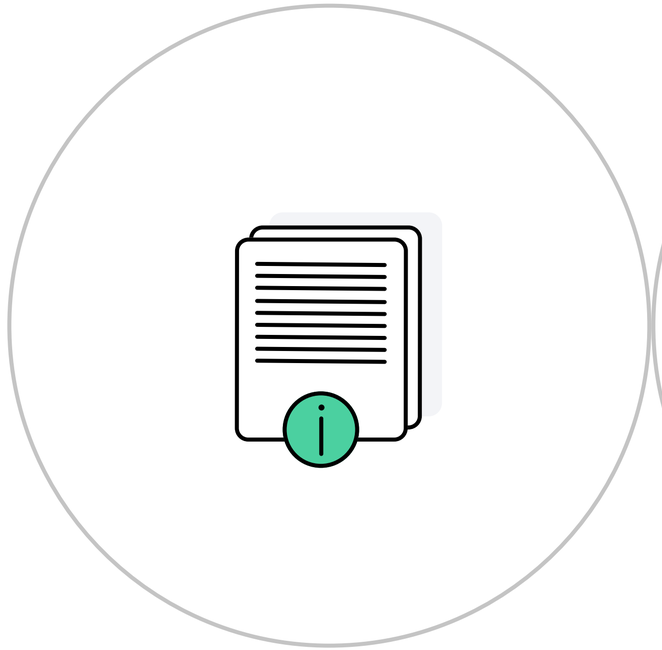
Что нужно учитывать при пагинации

- Количество страниц, если есть навигация на произвольную страницу
- Есть ли следующая или предыдущая страница
- Переход на несуществующую страницу



Стандартный пагинатор Django

В Django уже всё это **учли**, и получился **класс** Paginator:
https://docs.djangoproject.com/en/3.2/topics/pagination/



Демонстрация: view

```
def articles_view(request):
paginator = Paginator(all_articles, 2)
current_page = request.GET.get('page', 1)
page = paginator.get_page(current_page)
context = {
    'page': page,
    'articles': page.object_list,
}
return render(request, 'demo/articles.html', context)
```

Демонстрация: шаблон

```
<!DOCTYPE html>
<html lang="ru">
<head>
 <title>Статьи</title>
</head>
<body>
 {% for article in articles %}
  <div>{{ article }}</div>
 {% endfor %}
 <p>Текущая страница: {{ page.number }}</p>
 {% if page.has_next %}
  <br>
  <a href="?page={{ page.next_page_number }}">Следующая страница</a>
 {% endif %}
 {% if page.has_previous %}
  <br>
  <a href="?page={{ page.previous_page_number }}">Предыдущая страница</a>
 {% endif %}
</body>
</html>
```