**Pagination in Django:**

Pagination is the process of breaking large chunks of data up across multiple, discrete web pages. Rather than dumping all the data to the user, you can define the number of individual records you want to be displayed per page and then send back the data that corresponds to the page requested by the user.

The advantage of using this type of technique is that it improves the user experience, especially when there are thousands of records to be retrieved. Implementing pagination in Django is fairly easy as Django provides a [Paginator](https://docs.djangoproject.com/en/4.0/ref/paginator/#paginator-class) class from which you can use to group content onto different pages.

## **Objectives**

By the end of this article, you will be able to:

1. Explain what pagination is and why you may want to use it.
2. Work with Django's Paginator class and Page objects.
3. Implement pagination in Django with function and class-based views.

## **Django Constructs**

When implementing pagination in Django, rather than re-inventing the logic required for pagination, you'll work with the following constructs:

1. [Paginator](https://docs.djangoproject.com/en/4.0/ref/paginator/#paginator-class) - splits a Django QuerySet or list into chunks of Page objects.
2. [Page](https://docs.djangoproject.com/en/4.0/ref/paginator/#page-class) - holds the actual paginated data along with pagination metadata

**from** **django.core.paginator** **import** Paginator  
**from django.core.paginator import** Paginator, EmptyPage, PageNotAnInteger

**Syntax :**

p = Paginator(list\_of\_objects, no\_of\_objects\_per\_page)

The first argument is the list of objects which will be distributed over pages. The second argument denotes the number of objects that will be displayed on each page. These two arguments are required.

The Paginator class has the following attributes:

1. count - total number of objects
2. num\_pages - total number of pages
3. page\_range - range iterator of page numbers

The Page object has several attributes and methods that can be used while constructing your template:

1. number - shows the page number for a given page
2. paginator - displays the associated Paginator object
3. has\_next() - returns True if there's a next page
4. has\_previous() - - returns True if there's a previous page
5. next\_page\_number() - returns the number of the next page
6. previous\_page\_number() - returns the number of the previous page

## **Function-based Views**

Next, let's look at how to work with pagination in function-based views:

**from** **django.core.paginator** **import** Paginator, EmptyPage, PageNotAnInteger

**from** **django.shortcuts** **import** render

**from** **.** models **import** **Employee**

**def** index(request):

object\_list = Employee.objects.all()

page\_num = request.GET.get('page', 1)

paginator = Paginator(object\_list, 6) *# 6 employees per page*

**try**:

page\_obj = paginator.page(page\_num)

**except** PageNotAnInteger:

*# if page is not an integer, deliver the first page*

page\_obj = paginator.page(1)

**except** EmptyPage:

*# if the page is out of range, deliver the last page*

page\_obj = paginator.page(paginator.num\_pages)

**return** render(request, 'index.html', {'page\_obj': page\_obj})

Here, we:

1. Defined a page\_num variable from the URL.
2. Instantiated the Paginator class passing it the required parameters, the employees QuerySet and the number of employees to be included on each page.

Generated a page object called page\_obj, which contains the paginated employee data along with metadata for navigating to the previous and next pages.

**Class-based Views**

Example of implementing pagination in a class-based view:

**from** **django.views.generic** **import** ListView

**from** **.** models **import** **Employee**

**class** **Index**(ListView):

model = Employee

context\_object\_name = 'employees'

paginate\_by = 6

template\_name = 'index.html'

*index.html*:

<!doctype html>

<**html** lang="en">

<**head**>

<**meta** charset="utf-8">

<**meta** name="viewport" content="width=device-width, initial-scale=1">

<**link** href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css">

<**title**>Pagination in Django</**title**>

</**head**>

<**body**>

<**div** class="container">

<**h1** class="text-center">List of Employees</**h1**>

<**hr**>

<**ul** class="list-group list-group-flush">

{% for employee in page\_obj %}

<**li** class="list-group-item">{{ employee }}</**li**>

{% endfor %}

</**ul**>

<**br**><**hr**>

<**div**>

<**span**>

{% if page\_obj.has\_previous %}

<**a** href="?page={{ page\_obj.previous\_page\_number }}">Previous</**a**>

{% endif %}

<**span**>

Page {{ page\_obj.number }} of {{ page\_obj.paginator.num\_pages }}.

</**span**>

{% if page\_obj.has\_next %}

<**a** href="?page={{ page\_obj.next\_page\_number }}">Next</**a**>

{% endif %}

</**span**>

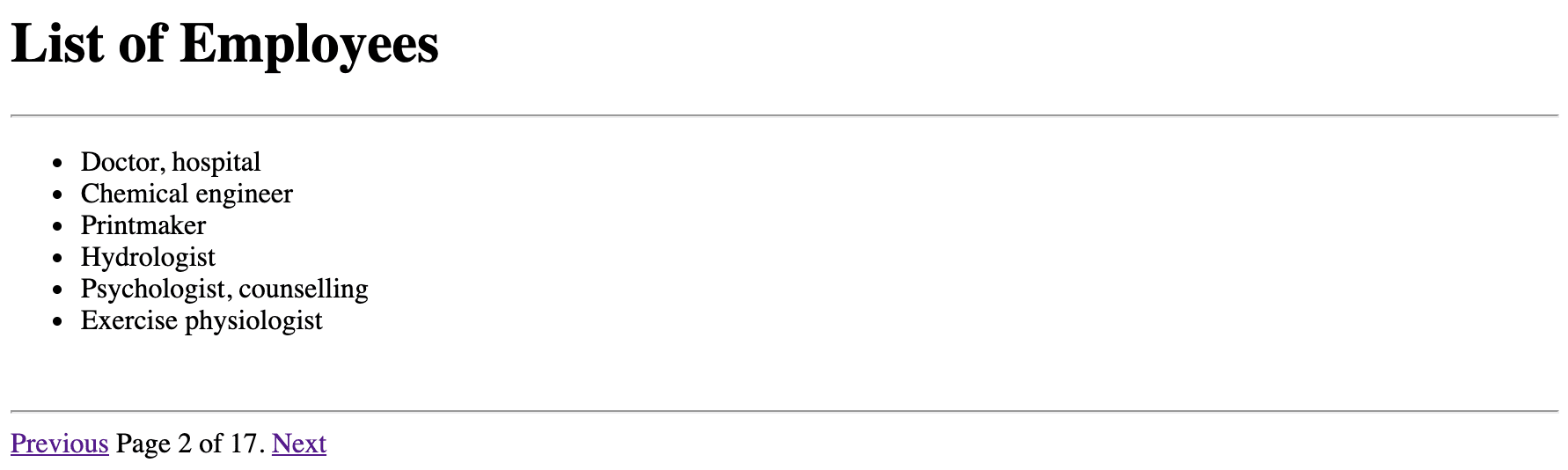
</**div**>

</**div**>

</**body**>

</**html**>

This is the first flavor implementing the pagination UI.



So, in this example, we have "Previous" and "Next" links that the end-user can click to move from page to page.