2) What is render()?
Combines a given template with a given context dictionary and returns an HttpResponse object with that rendered text.

Required Arguments:-

1) Request
   The request object used to generate this response.
2) template_name
   The full name of a template to use or sequence of template names. If a sequence is given, the first template that exists will be used.

Optional Arguments:-

Context :- A dictionary of values to add to the template context. By default, this is an empty dictionary. If a value in the dictionary is callable, the view will call it just before rendering the template.

3) Difference between {% extends %} and {% include %}
Extending allows you to replace blocks (e.g. "content") from a parent template instead of including parts to build the page (e.g. "header" and "footer"). This allows you to have a single template containing your complete layout and you only "insert" the content of the other template by replacing a block.

Extends in Template inheritance
Includes adds the whole template in your file.

4) write steps to create the custom tags and filter, give one working example of each.

The app should contain a templatetags directory at the same levels as models.py or views.py.
It should also contain a __init__.py file so it is treated as a package.
The custom tags can then be loaded in the template. For example if the file name is:
Tags_custom then in the template we can add {% load Tags_custom %}.

The app that contains the custom tag must be in INSTALLED_APPS in settings.
To be a valid tag library, the module must contain a module-level variable named register that is a template.Library instance, in which all the tags and filters are registered. So, near the top of your module, put the following:
register = template.Library()

Writing custom template filters:-
Custom filters are just Python functions that take one or two arguments:

- The value of the variable (input) – not necessarily a string.
- The value of the argument – this can have a default value, or be left out altogether.

For example, in the filter {{ var|foo:"bar" }}, the filter foo would be passed the variable var and the argument "bar".
Since the template language doesn't provide exception handling, any exception raised from a template filter will be exposed as a server error. Thus, filter functions should avoid raising exceptions if there is a reasonable fallback value to return. In case of input that represents a clear bug in a template, raising an exception may still be better than silent failure which hides the bug

Example : def lower(value):
"""Converts a string into all lowercase"""
return value.lower()
Registering custom filters
register.filter('lower', lower)
Or it can be registered with a decorator instead.
@register.filter

5. write steps for setting up the statics files like .css or .js file, give one working example.

Static files include stuff like CSS, JavaScript and images that you may want to serve alongside your site.
The STATICFILES_DIRS tuple tells Django where to look for static files that are not tied to a particular app. In this case, we just told Django to also look for static files in a folder called static in our root folder, not just in our apps.
Django also provides a mechanism for collecting static files into one place so that they can be served easily. Using the collectstatic command, Django looks for all static files in your apps and collects them wherever you told it to, i.e. the STATIC_ROOT. In our case, we are telling Django that when we run python manage.py collectstatic, gather all static files into a folder called staticfiles in our project root directory. This feature is very handy for serving static files, especially in production settings.

Create a foder called static alongside your manage.py file
Create a css and js folder inside your static directory

6) What is pagination?
Pagination is a way to break up a lot of data into smaller chunks that is visually appealing and understandable.

7) DIRS: defines a list of directories where the engine should look for template source files, in search order.

APP_DIRS: tells whether the engine should look for templates inside installed applications. Each backend defines a conventional name for the subdirectory inside applications where its templates should be stored.

OPTIONS: contains backend-specific settings.

BACKEND: is a dotted Python path to a template engine class implementing Django's template backend API. The built-in backends are django.template.backends.django.DjangoTemplates and django.template.backends.jinja2.Jinja2.