```
def task_list(request):
    tasks = Task.objects.all()
    print(tasks)
    return render(request, 'tasks/task_list.html', {'tasks': tasks})
```

Function Definition:

- **def task_list(request):**: This defines a view function named **task_list** that takes an HTTP request (**request**) as its parameter.

Querying Tasks:

- **tasks = Task.objects.all()**: This line queries all the tasks from the database using the Django ORM (Object-Relational Mapper). **Task.objects** represents the manager for the **Task** model, and **.all()** retrieves all instances of the **Task** model.

render Function:

- The **render** function is a shortcut provided by Django to simplify the process of loading a template, rendering it with context data, and returning an **HttpResponse** object with the rendered content.
- **'tasks/task_list.html'**: The template file to render. This is a relative path, assuming the templates are organized in a directory named **tasks** within the application's templates directory
- **'tasks'**: This is a key in the dictionary. It's like a label.
- **tasks**: This is the value associated with the key **'tasks'**. It refers to a list of tasks that you've retrieved from your database.

So, when you render the template with **render(request, 'tasks/task_list.html', {'tasks': tasks})**, you're telling Django to pass this dictionary (**{'tasks': tasks}**) to the template **'tasks/task_list.html'**.

Now, inside your template (**'tasks/task_list.html'**), you can access this data using the key **'tasks'**

```
def task_detail(request, pk):
    task = get_object_or_404(Task, pk=pk)
    return render(request, 'tasks/task_detail.html', {'task': task})
```

Function Definition:

- **def task_detail(request, pk):**: This defines a view function named **task_detail** that takes two parameters: **request**, which represents the HTTP request, and **pk**, which is the primary key of the task to be displayed.

Querying the Task:

- **task = get_object_or_404(Task, pk=pk)**: This line retrieves a single task object from the database based on its primary key (**pk**).
- **get_object_or_404** is a shortcut function provided by Django. It tries to fetch the object from the database, and if it doesn't exist, it raises a 404 error, indicating that the requested resource was not found.

Rendering the Template:

- **return render(request, 'tasks/task_detail.html', {'task': task})**: This line renders the template named **'tasks/task_detail.html'** with the task object (**task**) passed as context data.
    - **request**: The HTTP request object.
    - **'tasks/task_detail.html'**: The template file to render. This is a relative path, assuming the templates are organized in a directory named **tasks** within the application's templates directory.
    - **{'task': task}**: A dictionary containing context data to be passed to the template. In this case, it includes the single task object retrieved from the database.

Urls.py

```
path('task/<int:pk>/', views.task_detail, name='task_detail'),
```

In this pattern, **<int:pk>** specifies that the URL should include an integer value, which will be captured and passed to the **task_detail** view function as an argument named **pk**.

So, when a user visits a URL like **/task/1/**, Django captures the value **1** from the URL and passes it as the **pk** argument to the **task_detail** view function. This allows the view function to retrieve the task with primary key **1** from the database and display its details.

```
from django import forms
from .models import Task
class TaskForm(forms.ModelForm):
    class Meta:
        model = Task
        fields = ['title', 'description', 'due_date', 'completed']


def task_create(request):
    if request.method == 'POST':
        form = TaskForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('task_list')
    else:
        form = TaskForm()
    return render(request, 'tasks/task_create.html', {'form': form})
```

- This is a Django form class named **TaskForm** which inherits from **forms.ModelForm**. This means it's a model form, designed to work with Django models.
- **Meta** inner class provides metadata about the form.
- **model = Task** specifies that this form is associated with the **Task** model. This allows Django to automatically create form fields based on the model fields.
- **fields = [...]** specifies which fields from the **Task** model should be included in the form.

->This view function **task_create** handles the creation of a new task.

->It first checks if the request method is POST, which indicates that form data has been submitted.

- **TaskForm**: This is the form class you defined earlier using Django's **ModelForm** class. It represents the structure and behavior of your form.
- **request.POST**: This is the data that you're passing to the form. In HTTP, when a form is submitted via the POST method, the form data is sent to the server as part of the request body. In Django, **request.POST** is a dictionary-like object containing all the form data submitted via POST request.

- It then checks if the form is valid (**form.is_valid()**). If it is, it saves the form data to the database using **form.save()** and redirects the user to the task list page (**redirect('task_list')**).
- If the request method is not POST (i.e., it's a GET request), it creates a blank form instance.
- It renders the template **'tasks/task_create.html'** and passes the form instance as context data.

In summary, the **TaskForm** class defines a form for creating new tasks, and the **task_create** view function handles the creation process, displaying the form to the user and processing the submitted data.

```
path('task/new/', views.task_create, name='task_create'),
```

- **path**: This is a function provided by Django for defining URL patterns. It takes three arguments: the URL pattern, the view function that should be called when the URL pattern is matched, and an optional name for the URL pattern.
- **'task/new/'**: This is the URL pattern itself. It specifies the URL that should be matched. In this case, it represents the URL for creating a new task.
- **views.task_create**: This is the view function that should be called when the URL pattern is matched. In Django, views are Python functions that handle HTTP requests and return HTTP responses. In this case, **views.task_create** refers to the **task_create** function defined in the **views.py** file of the application.
- **name='task_create'**: This is an optional argument that provides a name for the URL pattern. Naming URL patterns allows you to refer to them by name in your Django templates and view functions, instead of hardcoding URLs. It promotes better code organization and makes your code more maintainable.

```
def task_update(request, pk):
    task = get_object_or_404(Task, pk=pk)
    if request.method == 'POST':
        form = TaskForm(request.POST, instance=task)
        if form.is_valid():
            form.save()
            return redirect('task_list')
    else:
        form = TaskForm(instance=task)
return render(request, 'tasks/task_update.html', {'form': form, 'task': task})
```

1. **TaskForm(request.POST, instance=task)**: This part creates an instance of the **TaskForm** class, which is a Django form used for updating task details. Here's what each part does:
    - **TaskForm**: This is the form class defined in your **forms.py** file. It inherits from **forms.ModelForm** and is responsible for generating HTML forms based on the **Task** model.
    - **request.POST**: This parameter contains the data submitted via an HTTP POST request. When a user submits a form, the form data is sent to the server in the POST request. By passing **request.POST** to the form, we bind the submitted data to the form instance, allowing Django to process and validate it.
    - **instance=task**: This parameter binds the form to an existing task object (**task**). By providing an instance of the **Task** model, the form is initialized with the data from this task object. This means that the form fields will be pre-filled with the existing task details, allowing the user to edit them.

In summary, **form = TaskForm(request.POST, instance=task)** creates a form instance based on the **TaskForm** class, binds it to the POST data submitted by the user, and pre-fills the form fields with the details of the existing task object (**task**). This allows the user to edit the task details and submit the updated form for saving.

return render(request, 'tasks/task_update.html', {'form': form, 'task': task}): This line renders the 'task_update.html' template, passing the form and task objects as context data. The template typically contains the form for updating the task details, with the pre-filled form fields populated with the existing task data.

```
def task_delete(request, pk):
    task = get_object_or_404(Task, pk=pk)
    if request.method == 'POST':
        task.delete()
        return redirect('task_list')  # Redirect to the task list page after deleting the task
    return render(request, 'tasks/task_delete.html', {'task': task})
```

1. This return statement executes when the HTTP request method is POST, indicating that the user has submitted a form to confirm the deletion of a task. Inside this block, the task object is deleted from the database using the **delete()** method, and then the user is redirected to the task list page (**'task_list'**). This return statement terminates the execution of the view function and immediately returns an HTTP redirect response.

2. This return statement is outside the conditional block. It executes when the HTTP request method is not POST, which typically occurs when the user initially accesses the page to confirm the deletion of a task. In this case, the view renders the 'task_delete.html' template, passing the task object as context data. This return statement generates an HTTP response containing the rendered HTML page.

In summary, the first return statement handles the case where the user confirms the task deletion and triggers an HTTP redirect response, while the second return statement handles the initial rendering of the confirmation page.