In an empty directory called walk_thru_graphic

```
(my_dj) g_joss ➡️ ~/Desktop/soft_pro/django_sandbox/walk_thru_graphic
🌀 ls
(my_dj) g_joss ➡️ ~/Desktop/soft_pro/django_sandbox/walk_thru_graphic
🌀 mkdir django-gunicorn-nginx; django-admin startproject project django-gunicorn-nginx/
(my_dj) g_joss ➡️ ~/Desktop/soft_pro/django_sandbox/walk_thru_graphic
🌀 ls
django-gunicorn-nginx
(my_dj) g_joss ➡️ ~/Desktop/soft_pro/django_sandbox/walk_thru_graphic
🌀 tree django-gunicorn-nginx
django-gunicorn-nginx
├── manage.py
└── project
    ├── __init__.py
    ├── asgi.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py

1 directory, 6 files
```

Run start project command in a newly created directory called django-gunicorn-nginx

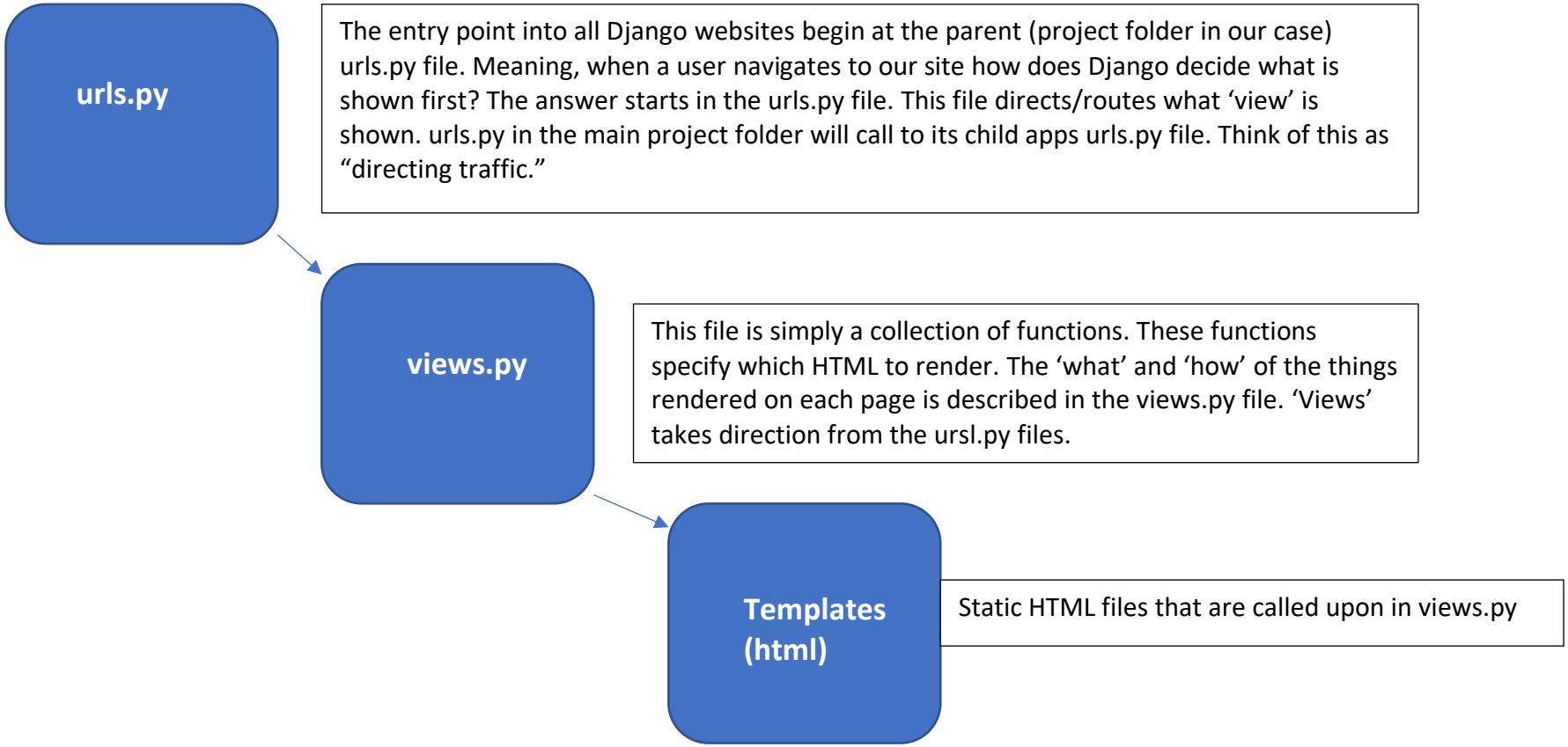
```
🌀 cd django-gunicorn-nginx; django-admin startapp myapp
(my_dj) g_joss ➡️ ~/Desktop/soft_pro/django_sandbox/walk_thru_graphic/django-gunicorn-nginx
🌀 tree
.
├── manage.py
├── myapp
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
└── project
    ├── __init__.py
    ├── asgi.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py

3 directories, 13 files
```

In Django jargon, Projects consist of apps. A whole complex app could be written within the main project folder without ever using the startapp command, but this would be an unwise design pattern. It is good practice a write a code base that is extensible and modular. The implementation of this idea with the Django framework would be to delegate functionality to separate self-contained apps. **<u>This is how we are going to integrate everyone's code</u>**

```
# Application definition
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'myapp'
]
```

This is a snippet from project/settings.py. Link the newly created app with the parent project

Describing the interplay between 3 important Django concepts:
urls.py file → views.py file → templates (html files)



```
(my_dj) g_joss ➡ ~/Desktop/soft_pro/django_sandbox/walk_thru_graphic/django-gunicorn-nginx
mkdir -vp myapp/templates
```

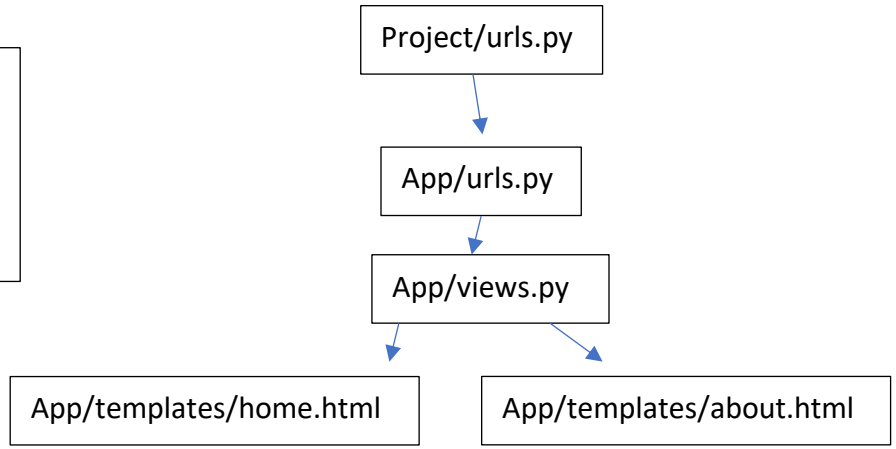
In the one and only application of this project, create a new directory called templates and populate this directory with two very basic HTML files

```
(my_dj) g_joss ➡ ~/Desktop/soft_pro/django_sandbox/walk_thru_graphic/django-gunicorn-nginx
cat > myapp/templates/home.html
<!DOCTYPE html>
<html lang="en-US">
<head>
  <meta charset="utf-8">
  <title>My secure app</title>
</head>
<body>
  <p>Now this is some sweet HTML!</p>
  <p><span >Now this is some sweet HTML!</span></p>
</body>
</html>
```

```
(my_dj) g_joss ➡ ~/Desktop/soft_pro/django_sandbox/walk_thru_graphic/django-gunicorn-nginx
cat > myapp/templates/about.html
<!DOCTYPE html>
<head>
  <meta charset="utf-8">
  <title>an abouts page</title>
  <meta name="description" content="">
  <link rel="stylesheet" href="">
</head>
<body>
  <h1>here is the story</h1>
  <script src=""></script>
</body>
</html>
```

I want to show these to HTML files:

1. What to put in the project's urls.py file that references the application's urls.py? Based on what you URL (screen) is being routed the urls.py file says which 'view' should be rendered
2. What to put in views.py file? "when this view is called upon what HTML should I render?"



project > urls.py > ...

```
1   from django.contrib import admin
1   from django.urls import path, include
2
3   urlpatterns = [
4       path('admin/', admin.site.urls),
5       path("", include("myapp.urls")),
6   ]
```

Line 5 in project/urls.py says whenever the path empty string (meaning the entry point of the site occurs return include("myapp.urls"). This is passing the call down the line to the child app.

Notice the import statement at the top of the file. This calls in myapp/views.py into the current namespace. To summarize line 4, when the sites homepage is accessed return the view called index. This means there is a function called index in the views file. Regarding the 3rd line, when /about is navigated to by a user return this view called about. The third argument name="about" will be talked about soon

myapp > urls.py > ...

```
8   from django.urls import path
7   from . import views
6
5   urlpatterns = [
4       path("", views.index),
3       path("about/", views.about, name="about"),
2   ]
```

myapp > views.py > ...

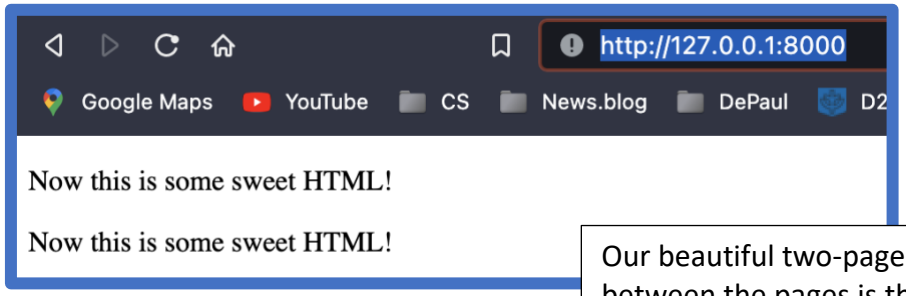
```
1   from django.shortcuts import render
1
2   # Create your views here.
3   def index(request):
4       return render(request, "home.html")
5
6   def about(request):
7       return render(request, "about.html")
8
```

myapp > templates > home.html > ...

```
1   <!DOCTYPE html>
1   <html lang="en-US">
2   <head>
3       <meta charset="utf-8">
4       <title>My secure app</title>
5   </head>
6   <body>
7       <p>Now this is some sweet HTML!</p>
8       <p><span >Now this is some sweet HTML!</span></p>
9   </body>
10  </html>
```

myapp > templates > about.html > ...

```
1   <!DOCTYPE html>
1   <head>
2       <meta charset="utf-8">
3       <title>an abouts page</title>
4       <meta name="description" content="">
5       <link rel="stylesheet" href="">
6   </head>
7   <body>
8       <h1>here is the story</h1>
9       <script src=""></script>
10  </body>
11  </html>
12
```



Our beautiful two-page site. The only way to navigate between the pages is through typing in the URL box. Let's change that

The snippet to the right should go somewhere between the body tags of index.html. The value of the href attribute is an example of the Django template language (DTL)

```
<button type="submit" >
  <a href="{% url 'about' %}">about page</a>
</button>
```

Next objective: passing user entered data to a new page.

Recipe:

1. Create a new html page that we are sending data to
2. Create a submit button on our landing page that accepts text
3. Add a new path object to the url patterns list in the child app's urls.py file
4. Create a function in views.py that retrieves the user input data and passes this input data along with the rendered HTML file

myapp > templates > signup.html > body

```
1  <h1> you signed up for the email list!!!</h1>
2  <script src=""></script>
3    <div class="container">
4      <h2>Added
5        <span style="text-decoration: underline;">
6          {{ email_placing }}</span>
7      to mailing list</h2>
8    </div>
```

Step 1 :
This is the page we will be sending user enter data to. Notice on line 6 in between the span tags there is a DTL invocation. When this page is rendered that email variable will be populated.

myapp > templates > home.html > html > body

```
1        <br/>
2        <br/>
3        <h2>Join our mailing list:</h2>
4        <form action="{% url 'signup' %}">
5        <label for="email">
6          Enter your email:
7        </label>
8        <input type="email" name="an_email"/>
9        <button type="submit">
10         Sign Up
11       </button>
12       </form>
```

Step 2 :
Notice the form tag on line 4. The attribute action takes the value of a path. The value of the action attribute is a DTL invocation. This means when this form is submitted send a data to that page. Notice on line 8 the name attribute within the input tag. This is how the input will be referred to later on.

myapp > urls.py > ...

```
7  from django.urls import path
6  from . import views
5
4  urlpatterns = [
3      path("", views.index),
2      path("about/", views.about, name="about"),
1      path("signup/", views.signup, name="signup")
8  ]
```

Step 3 :
The third argument to the path function specifies how to invoke this page with DTL.

Step 4 :
When the signup page is rendered, it expects a variable called an_email to be retrievable. Passing the data to the new rendered page is as easy as appending a dictionary on to the render function.

myapp > views.py > ...

```
1  def signup(request):
2      the_email = request.GET.get('an_email')
3      return render(request, 'signup.html', {'email_placing':the_email})
```