**Django**

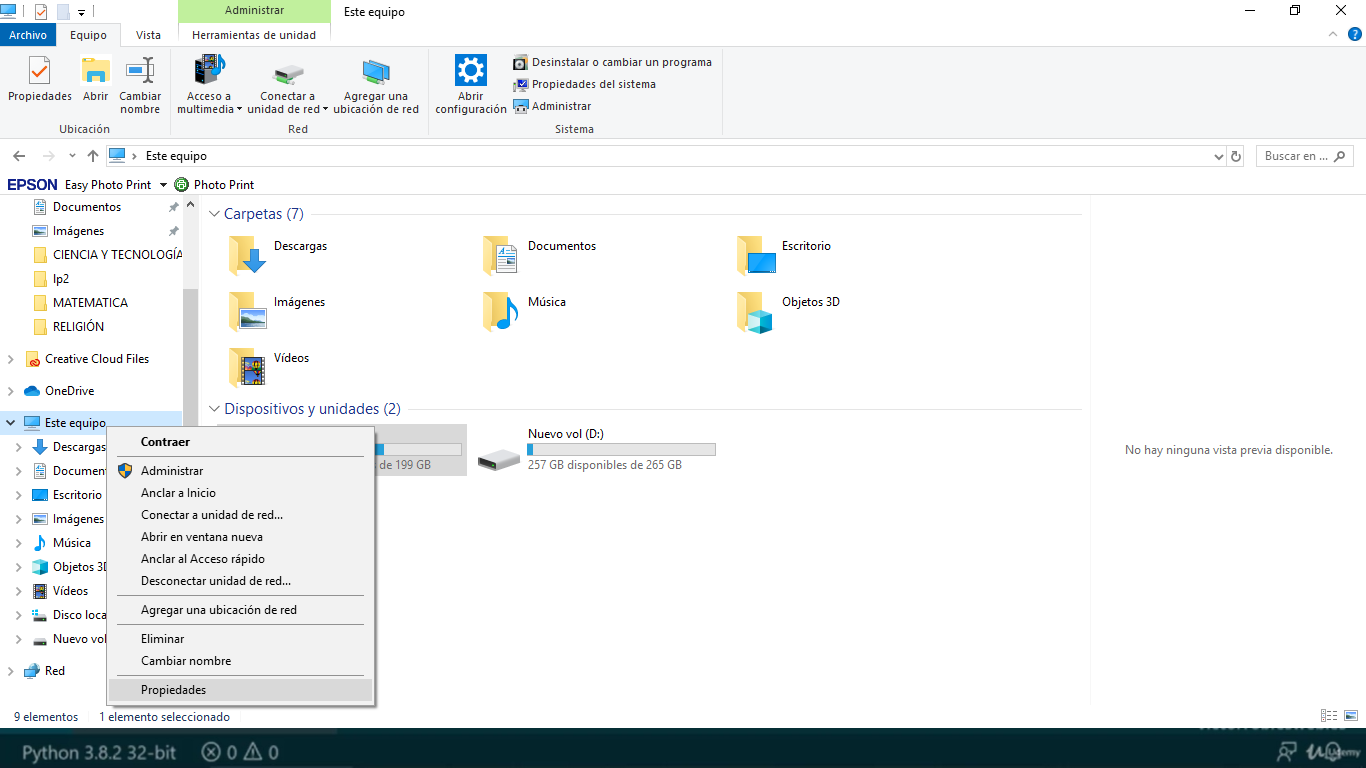
1. **INSTALACIÓN Y CREACIÓN DE PROYECTOS**
   1. **Introducción**

Django es un Framework web gratuito y de código abierto escrito en python

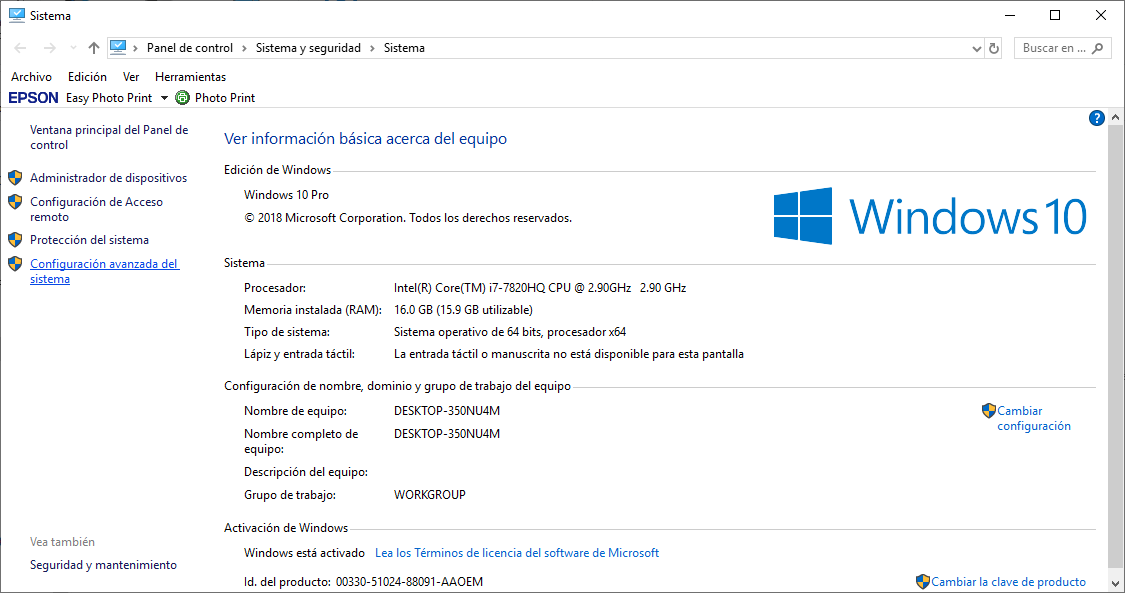
| **Recordemos que**:  Un **Framework**, es un marco de trabajo formado por un conjunto de:   * Herramientas * Librerías * Buenas Prácticas |
| --- |

Con Django podemos:

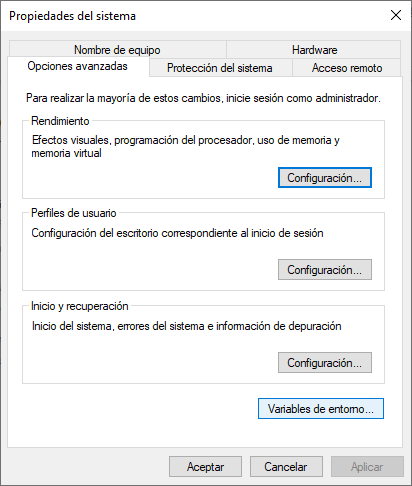
* Crear aplicaciones web
* Reutilizar código
* Realizar tareas repetitivas y pesadas durante la creación de un sitio web.
  1. **Arquitectura**
  2. Modelo - Vista - Controlador
  3. **Previo a la Instalación (Opcional)**
  4. Clic derecho en Mi Equipo



* 1. Seleccionar: Configuración avanzada



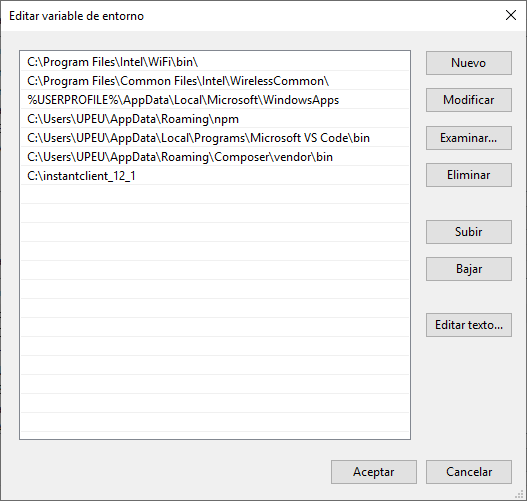
* 1. Variables de entorno



* 1. Seleccionamos **path** y **editar**



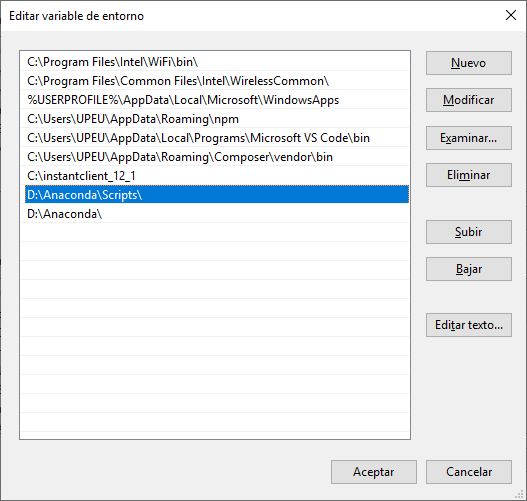
* 1. Hacer clic en el botón Nuevo



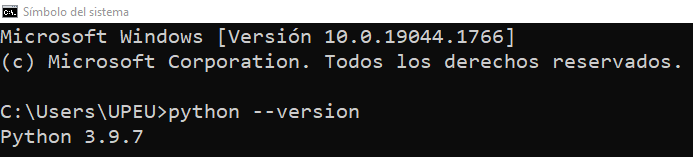
* 1. Ir a la carpeta donde está instalado Python (Anaconda), copiamos la dirección y pegamos la ubicación

Ejemplo:

* D:\Anaconda\Scripts\
* D:\Anaconda\

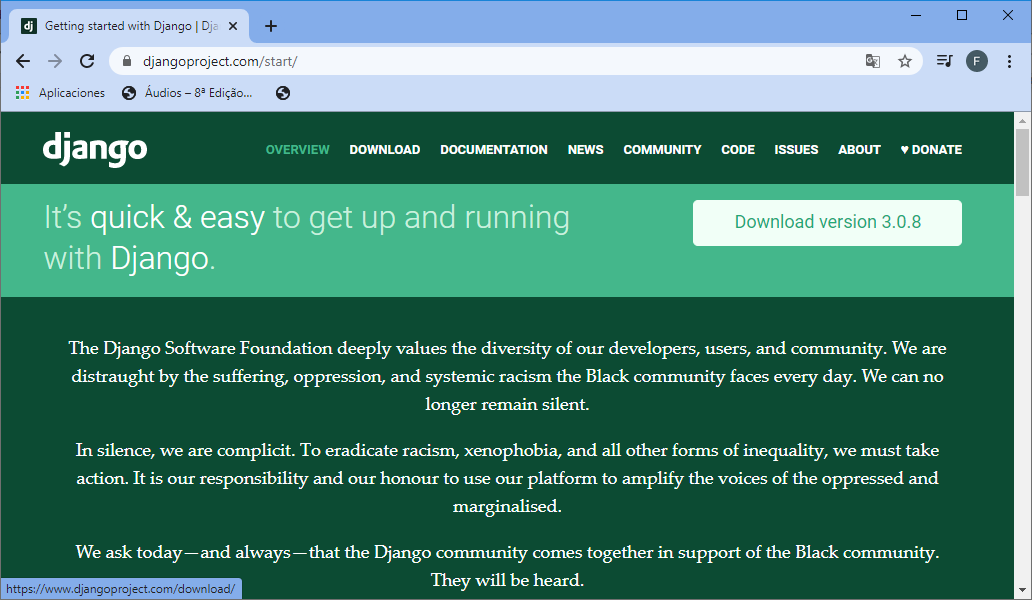


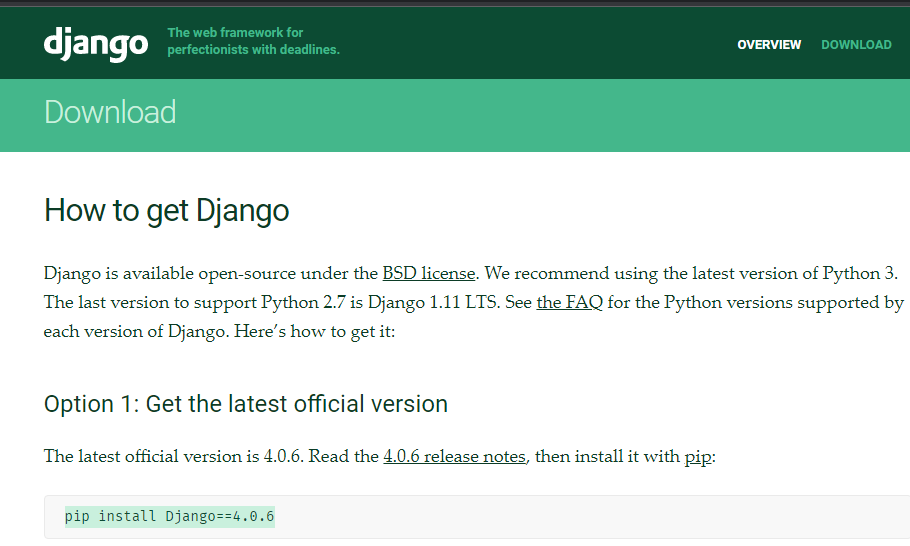
* 1. Luego aceptar, aceptar y aceptar.
  2. Ahora desde cmd probamos python



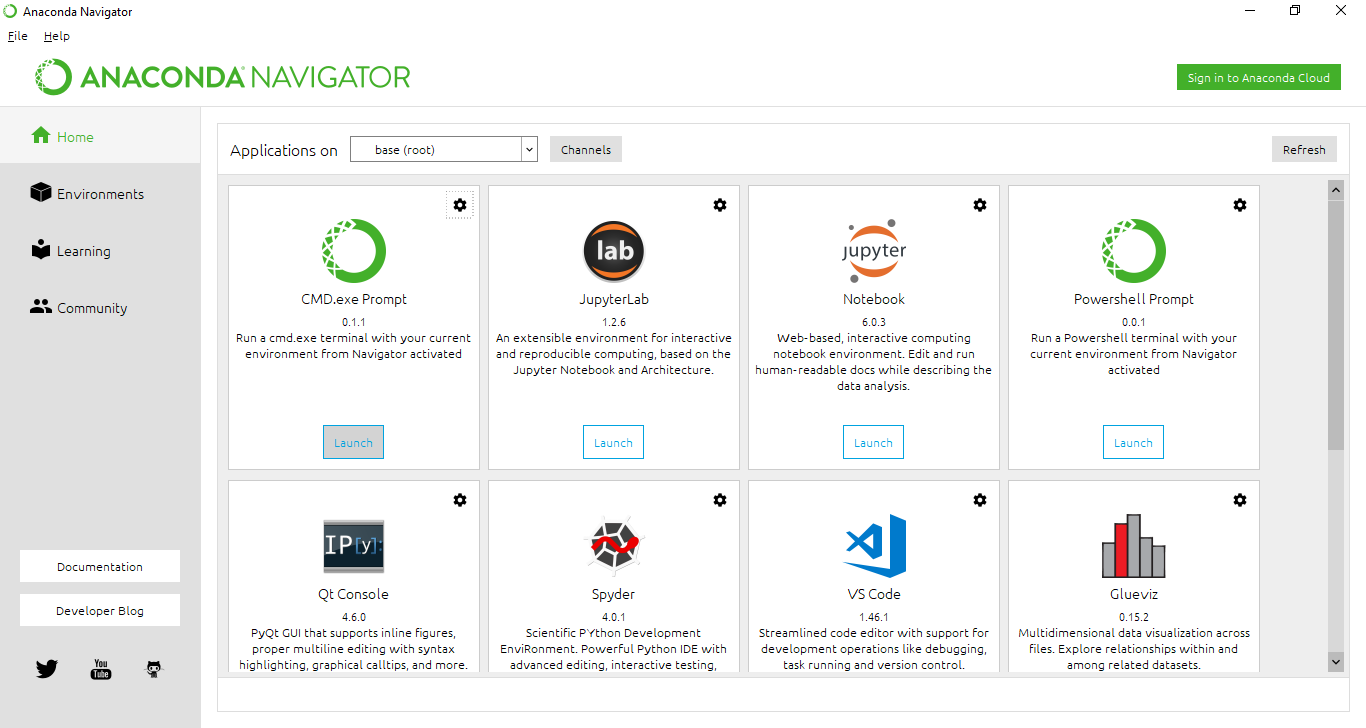
* 1. **Instalación**

Ir a la página: <https://www.djangoproject.com/>

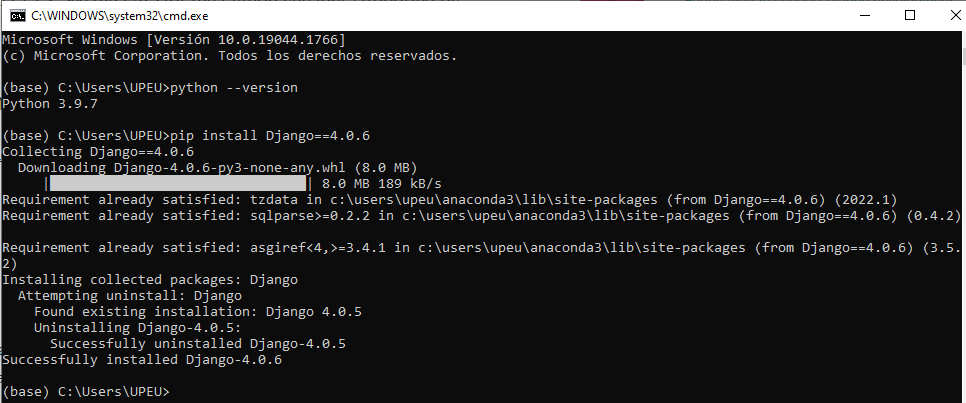




Abrimos la consola de python: **CMD.exe Prompt**



Digitamos: pip install Django==4.0.6

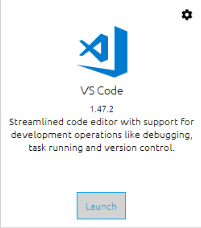


Para probar que versión de django tenemos instalado: **Python -m django –version**

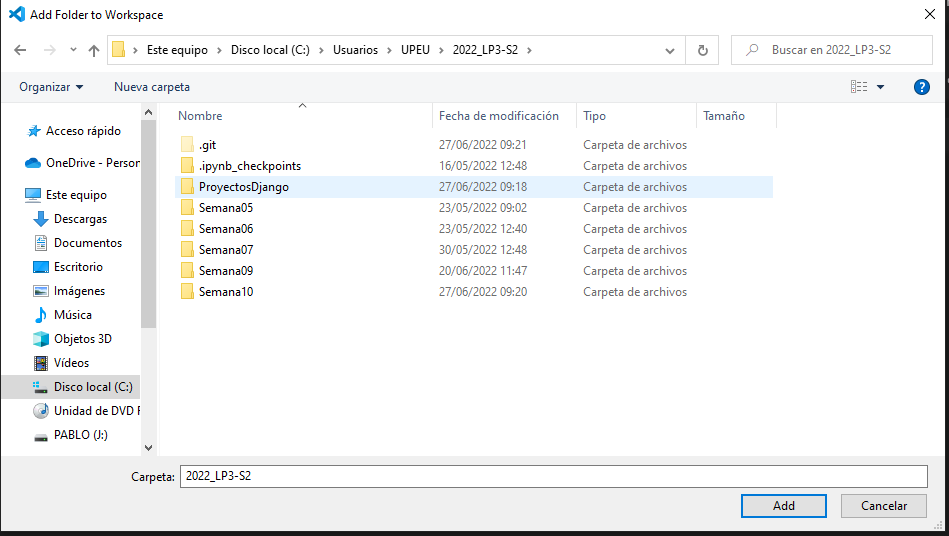
****

* 1. **Creación de mi primer proyecto**

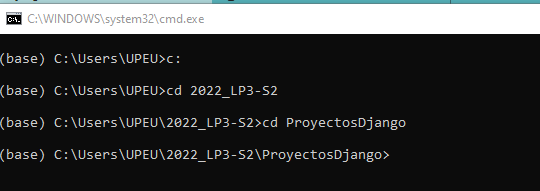
1. Ingresamos a VS Code



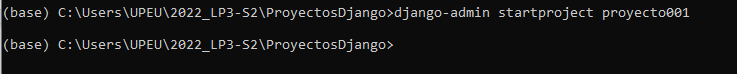
1. Creamos una carpeta llamada: **ProyectosDjango** y hacemos clic en **Add**



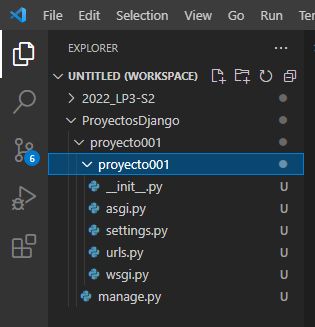
1. Desde la consola nos ubicamos en la carpeta creada



1. Creamos un proyecto Django, llamado: **proyecto001**, utilizando el comando: **django-admin startproyect proyecto001**



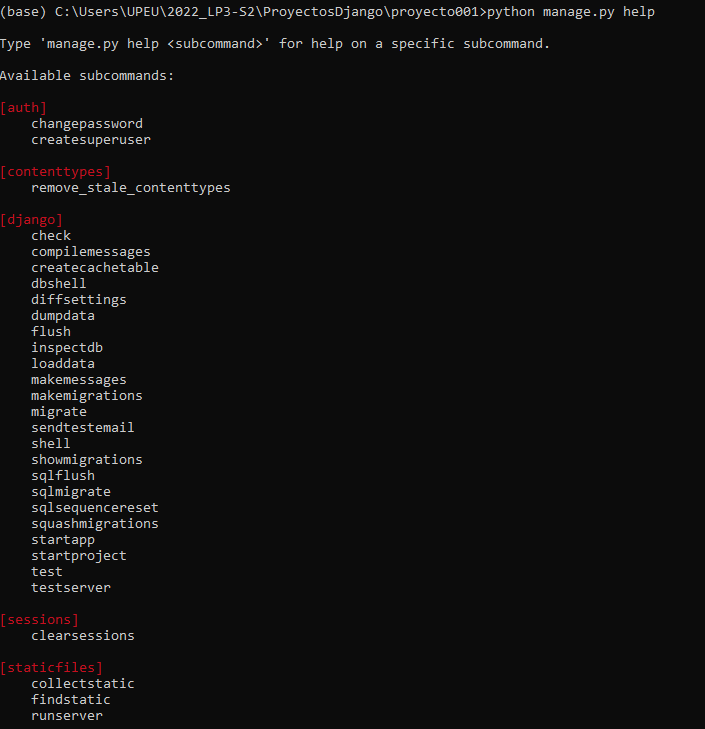
1. Podemos visualizar en VS Code los archivos generados



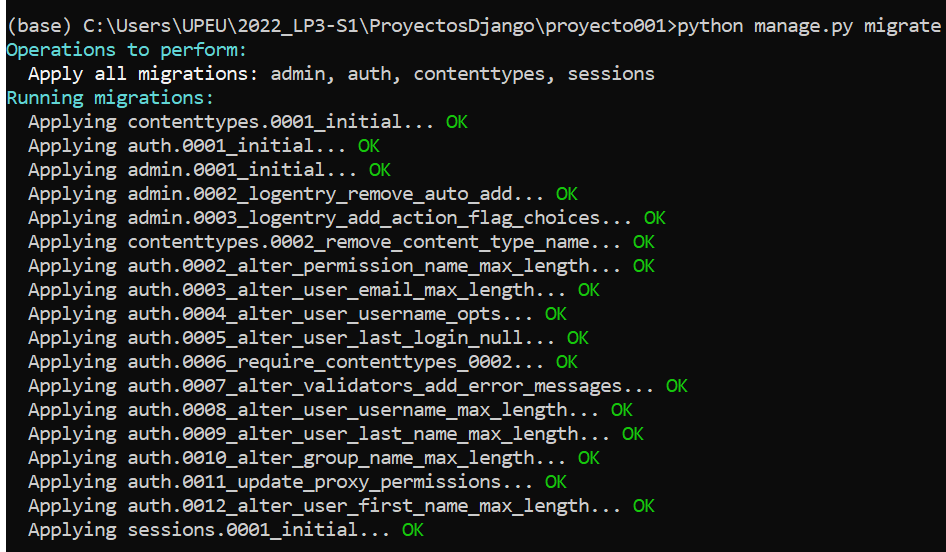
1. Al ingresar al proyecto,



1. Podemos visualizar todos los comandos que podemos ejecutar con manage.py: **python manage.py help**



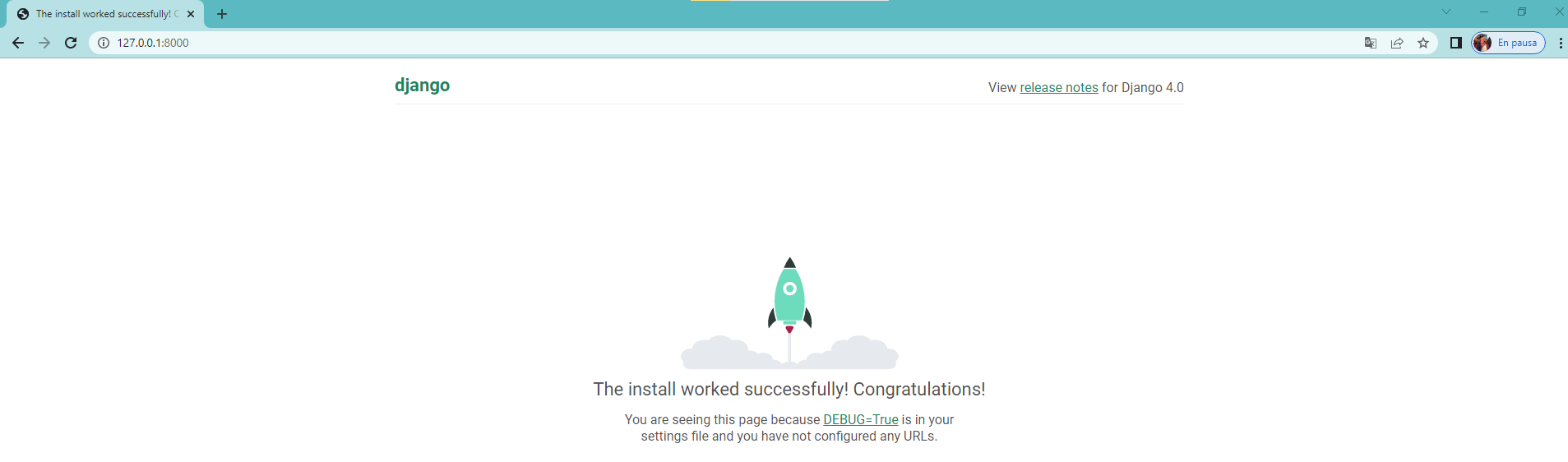
1. Ahora, para activar nuestro proyecto, entramos a la carpeta Proyecto001:**python manage.py migrate**

****

1. Para ejecutar nuestro proyecto en el servidor web(Por defecto) hacemos:**python manage.py runserver**

****

1. **Si copiamos el url en el navegador podemos observar**

****

**En resumen:**

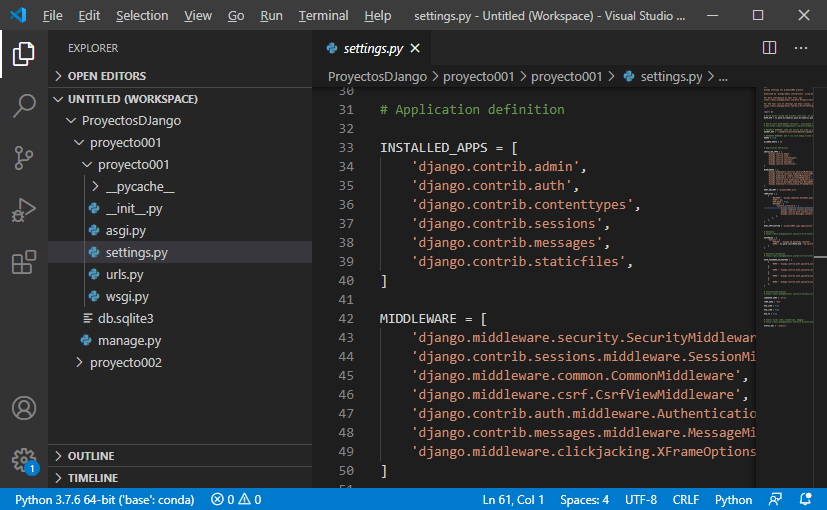
| * **django-admin startproject proyecto001** Para crear proyecto: proyecto001 * **python manage.py migrate** Para activar el proyecto: proyecto001 * **python manage.py runserver** Para ejecutar el proyecto en el servidor web |
| --- |

1. **CREACIÓN DE APPS**
   1. **Introducción**

Es un paquete que tendrá ciertas funcionalidades y puede ser reutilizado. Ejemplo: Categoria, Entrada, Salida, Blog, Tienda.

Lo ideal es tener un app para cada entidad.

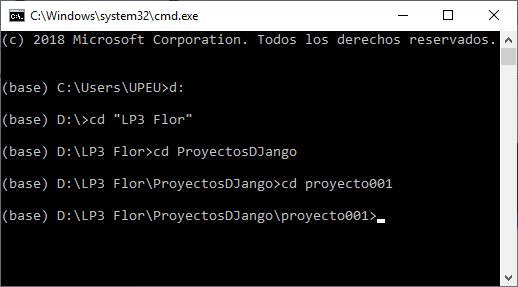
* 1. **En settings.py**

****

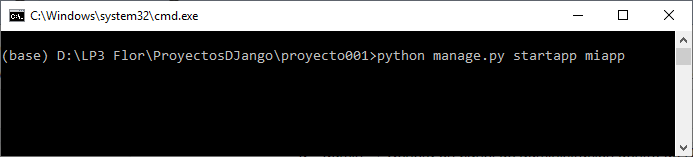
**Ejemplo:**

admin → Genera un panel de administración dentro de DJango

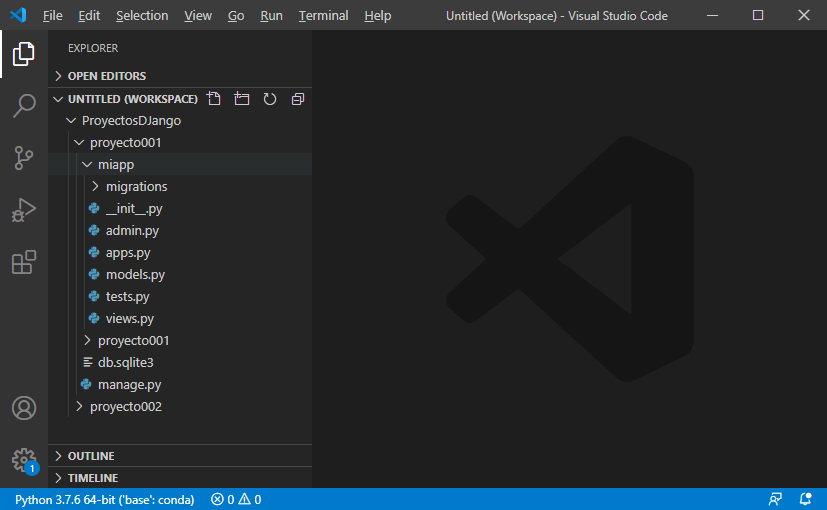
* 1. **Para crear un app**
  2. Por consola voy a la carpeta de mi proyecto



* 1. Para iniciar (Crear) un app: ***python manage.py startapp miapp***



* 1. Podemos visualizar en VS Code



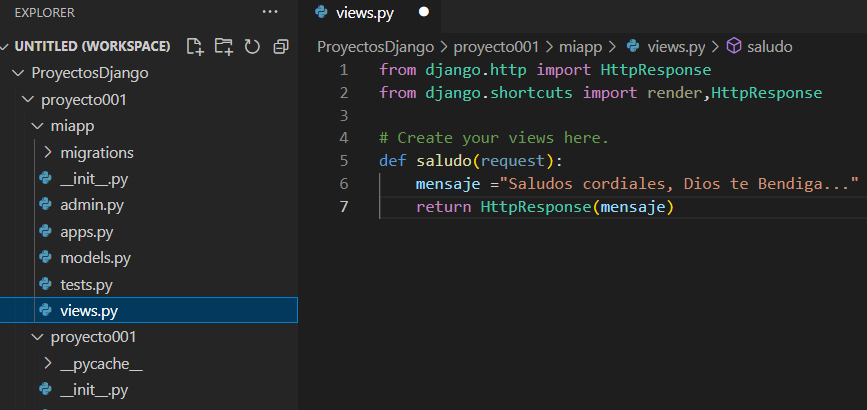
1. **ROUTING Y VISTAS**
   1. **Introducción**

Considerar:

**MVC**: Modelo Vista Control

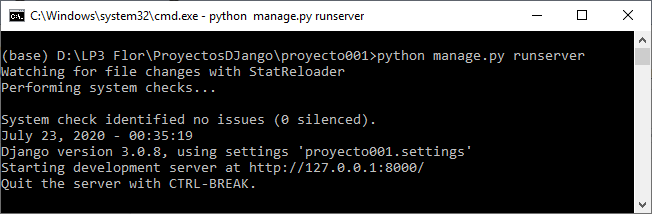
**MVT**: Modelo Template Vista

* 1. **En views.py**

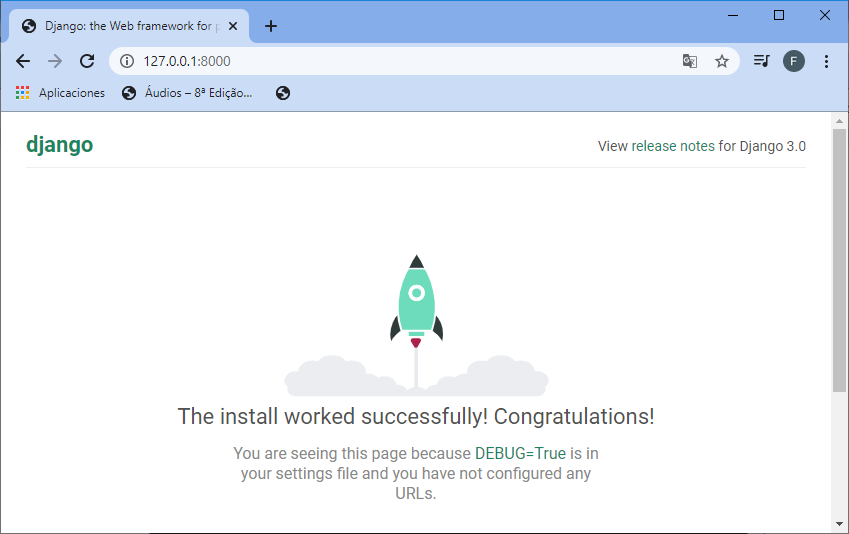
****

| **from django.http import HttpResponse**  **from django.shortcuts import render,HttpResponse**  **# Create your views here.**  **def saludo(request):**  **mensaje ="Saludos cordiales, Dios te Bendiga..."**  **return HttpResponse(mensaje)** |
| --- |

* 1. **Inicializamos el servidor**



* 1. **Al visualizar la página en el navegador, no se observan cambios**

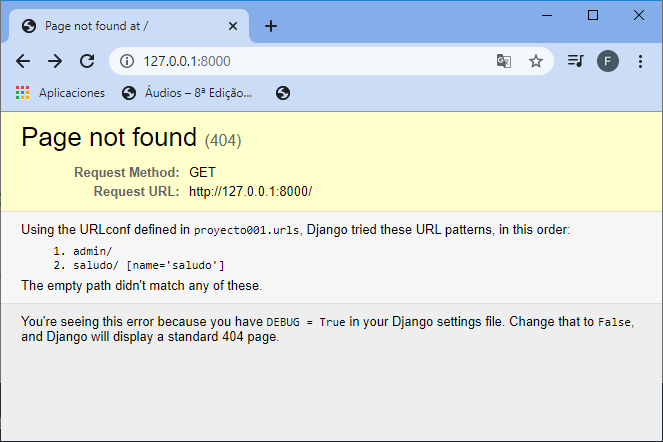


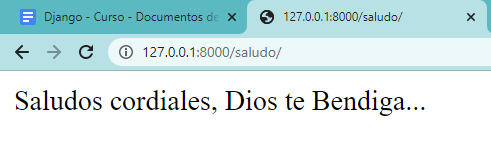
* 1. **En urls.py**



| from django.contrib import admin  from django.urls import path  ***from miapp import views***  urlpatterns = [  path('admin/', admin.site.urls),  ***path('saludo/',views.saludo, name = "saludo"),***  **]** |
| --- |

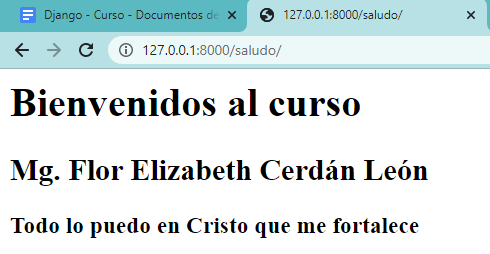
* 1. **Al cargar el proyecto**

****

****

* 1. **Puedo utilizar etiquetas html para dar formato al mensaje:**

| **from django.http import HttpResponse**  **from django.shortcuts import render,HttpResponse**  **# Create your views here.**  **def saludo(request):**  **mensaje ="""**  **<h1>Bienvenidos al curso</h1>**  **<h2>Mg. Flor Elizabeth Cerdán León</h2>**  **<h3>Todo lo puedo en Cristo que me fortalece</h3>**  **"""**  **return HttpResponse(mensaje)** |
| --- |

****

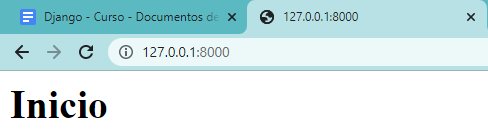
* 1. **Ahora crearemos otro método: index y lo vinculamos para que cargue por defecto.**

**En views.py**

| **from django.shortcuts import render, HttpResponse**  **# Create your views here.**  ***def index(request):***  ***mensaje="""***  ***<h1>Inicio</h1>***  ***"""***  ***return HttpResponse(mensaje)***    **def saludo(request):**  **mensaje = """**  **<h1>Bienvenidos</h1>**  **<h2>Flor Cerdán</h2>**  **<h3>Python....</h3>**  **"""**  **return HttpResponse(mensaje)** |
| --- |

**En urls.py**

| **from django.contrib import admin**  **from django.urls import path**  **from miapp import views**  **urlpatterns = [**  **path('admin/', admin.site.urls),**  **path('', views.index, name = "index"),**  **path('saludo/',views.saludo, name = "saludo"),** |
| --- |

****

* 1. **Si deseamos que el método index cargue adicionalmente en un URL llamado “inicio”**

| **import imp**  **from django.contrib import admin**  **from django.urls import path**  **from miapp import views**  **urlpatterns = [**  **path('admin/', admin.site.urls),**  **path('',views.index, name = "index"),**  **path('saludo/', views.saludo, name="saludo"),**  **path('inicio/', views.index, name = "inicio")**  **]** |
| --- |

* 1. **Si deseo crear un método que muestre el rango de números de [a,b], entonces:**

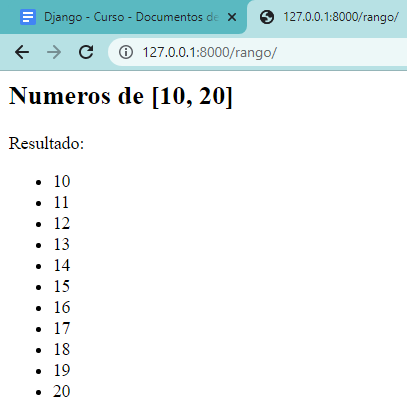
**En views:**

| **from django.shortcuts import render, HttpResponse**  **# Create your views here.**  **def index(request):**  **mensaje="""**  **<h1>Inicio</h1>**  **"""**  **return HttpResponse(mensaje)**  **def saludo(request):**  **mensaje = """**  **<h1>Bienvenidos</h1>**  **<h2>Flor Cerdán</h2>**  **<h3>Python....</h3>**  **"""**  **return HttpResponse(mensaje)**  **def rango(request):**  **a = 10**  **b = 20**  **resultado = f"""**  **<h2> Numeros de [{a},{b}] </h2>**  **Resultado: <br>**  **<ul>**  **"""**  **while a<=b:**  **resultado += f"<li> {a} </li>"**  **a+=1**  **resultado += "</ul"**  **return HttpResponse(resultado)** |
| --- |

**En urls.py**

| **from django.contrib import admin**  **from django.urls import path**  **from miapp import views**  **urlpatterns = [**  **path('admin/', admin.site.urls),**  **path('', views.index, name = "index"),**  **path('inicio/', views.index, name = "inicio"),**  **path('saludo/',views.saludo, name = "saludo"),**  **path('rango/',views.rango,name="rango"),**  **]** |
| --- |

**Al Cargar**

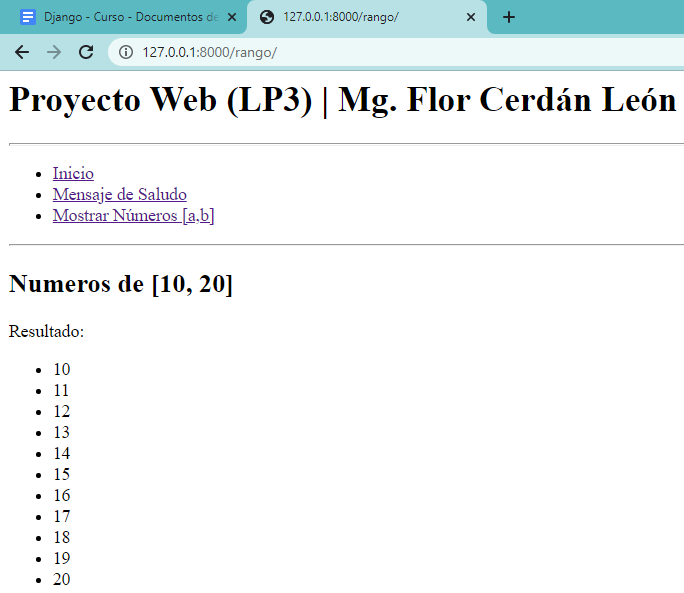
****

* 1. **Para navegar entre rutas**

**En views.py**

| **from django.shortcuts import render, HttpResponse**  **# Create your views here.**  **layout = """**  **<h1> Proyecto Web (LP3) | Flor Cerdán </h1>**  **<hr/>**  **<ul>**  **<li>**  **<a href="/inicio"> Inicio</a>**  **</li>**  **<li>**  **<a href="/saludo"> Mensaje de Saludo</a>**  **</li>**  **<li>**  **<a href="/rango"> Mostrar Números [a,b]</a>**  **</li>**  **</ul>**  **<hr/>**  **"""**  **def index(request):**  **mensaje="""**  **<h1>Inicio</h1>**  **"""**  **return HttpResponse(layout + mensaje)**  **def saludo(request):**  **mensaje = """**  **<h1>Bienvenidos</h1>**  **<h2>Flor Cerdán</h2>**  **<h3>Python....</h3>**  **"""**  **return HttpResponse(layout + mensaje)**  **def rango(request):**  **a = 10**  **b = 20**  **resultado = f"""**  **<h2> Números de [{a},{b}] </h2>**  **Resultado: <br>**  **<ul>**  **"""**    **while a<=b:**  **resultado += f"<li> {a} </li>"**  **a+=1**  **resultado += "</ul"**  **return HttpResponse(layout + resultado)** |
| --- |

**Al cargar**

****

* 1. **Rutas con parámetros**

Si deseo enviar valores en las rutas de mi página entonces:

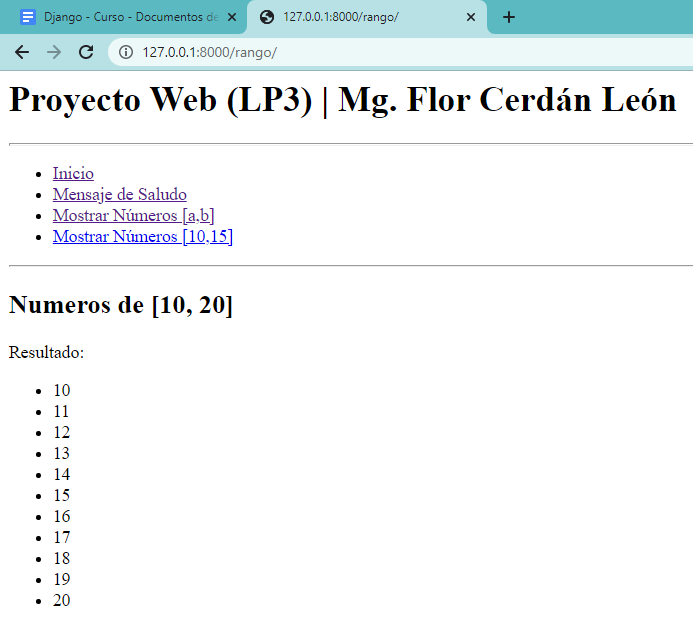
**views.py**

| **from django.http import HttpResponse**  **from django.shortcuts import render,HttpResponse**  **# Create your views here.**  **layout = """**  **<h1>Proyecto Web (LP3) | Mg. Flor Cerdán León</h1>**  **<hr/>**  **<ul>**  **<li>**  **<a href="/inicio">Inicio</a>**  **</li>**  **<li>**  **<a href="/saludo">Mensaje de Saludo</a>**  **</li>**  **<li>**  **<a href="/rango">Mostrar Números [a,b]</a>**  **</li>**  **<li>**  **<a href="/rango2/10/15">Mostrar Números [10,15]</a>**  **</li>**    **</ul>**  **<hr/>**  **"""**  **def index(request):**  **mensaje="""**  **<h1>Inicio</h1>**  **"""**  **return HttpResponse(layout + mensaje)**  **def saludo(request):**  **mensaje ="""**  **<h1>Bienvenidos al curso</h1>**  **<h2>Mg. Flor Elizabeth Cerdán León</h2>**  **<h3>Todo lo puedo en Cristo que me fortalece</h3>**  **"""**  **return HttpResponse(layout + mensaje)**  **def rango(request):**  **a=10**  **b=20**  **resultado = f"""**  **<h2> Números de [{a}, {b}] </h2>**  **Resultado: <br>**  **<ul>**  **"""**  **while a<=b:**  **resultado += f"<li> {a} </li>"**  **a+=1**  **resultado += "</ul>"**  **return HttpResponse(layout + resultado)**  **def rango2(request,a=0,b=100):**  **resultado = f"""**  **<h2> Números de [{a},{b}] </h2>**  **Resultado: <br>**  **<ul>**  **"""**  **while a<=b:**  **resultado += f"<li> {a} </li>"**  **a+=1**  **resultado += "</ul>"**  **return HttpResponse(layout + resultado)** |
| --- |

**urls.py**

| **from django.contrib import admin**  **from django.urls import path**  **from miapp import views**  **urlpatterns = [**  **path('admin/', admin.site.urls),**  **path('',views.index, name = "index"),**  **path('saludo/', views.saludo, name="saludo"),**  **path('inicio/', views.index, name = "inicio"),**  **path('rango/',views.rango, name = "rango"),**  **path('rango2/',views.rango2, name = "rango2"),**  **path('rango2/<int:a>',views.rango2, name = "rango2"),**  **path('rango2/<int:a>/<int:b>',views.rango2, name="rango2"),**  **]** |
| --- |

**Al Cargar**

****

* 1. **Si deseo configurar valores por defecto**

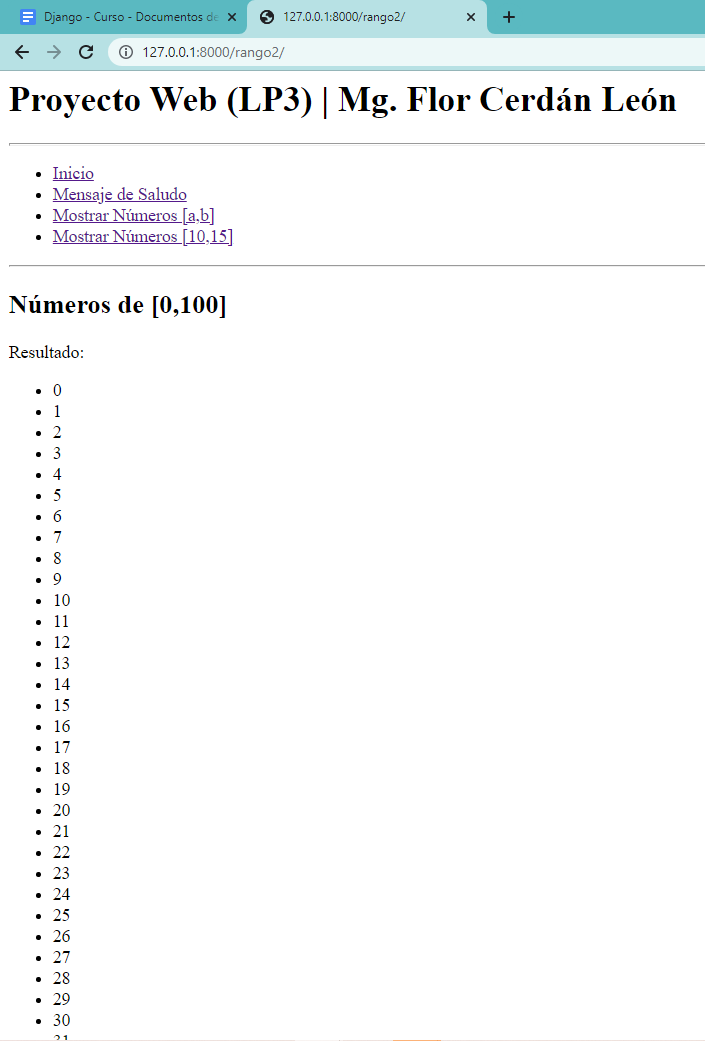
**En views.py**

| **…..**  **def rango2(request,a=0,b=100):**  **….** |
| --- |

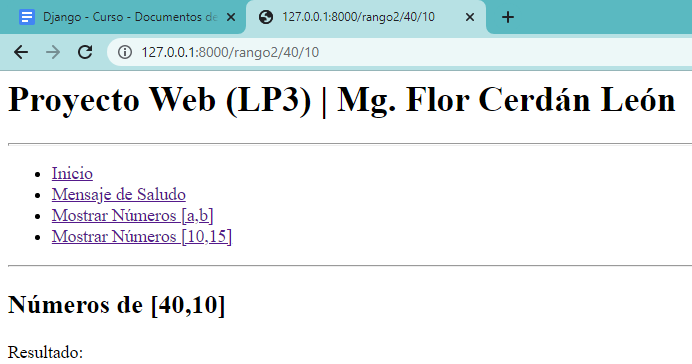
**En urls.py**

| **….**  **path('rango2/',views.rango2,name="rango2"),**  **path('rango2/<int:a>',views.rango2,name="rango2"),**  **path('rango2/<int:a>/<int:b>',views.rango2,name="rango2"),**  **….** |
| --- |

**Al Cargar**

****

* 1. **¿Que pasa si enviamos como parámetro a = 40, b = 10**

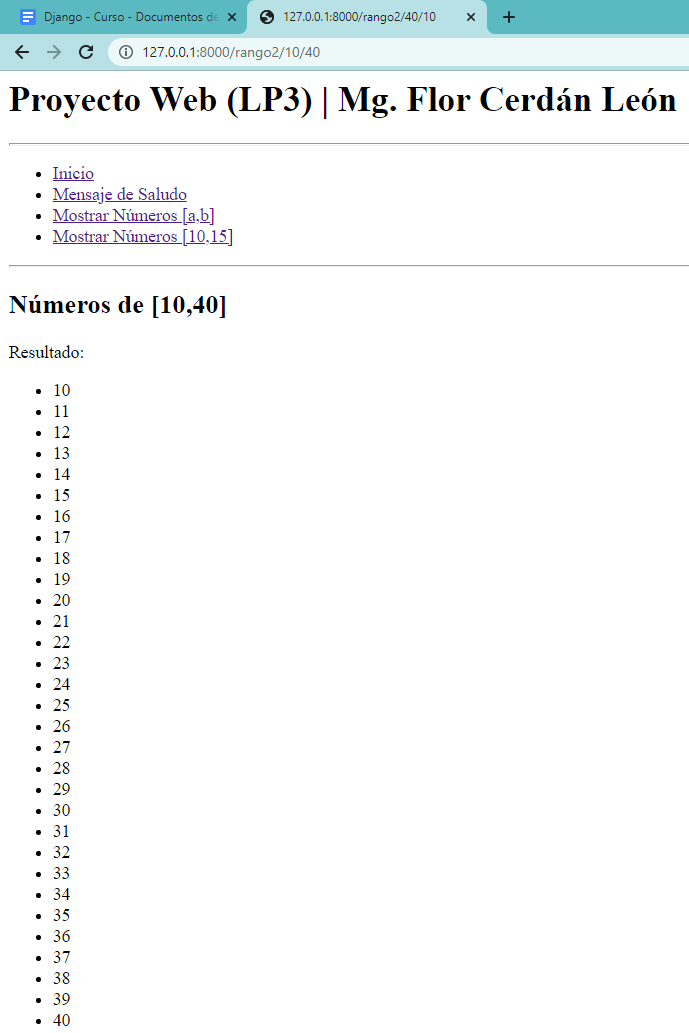
****

* 1. **Entonces, podemos utilizar “redirect”:**

**En views.py**

| **from django.shortcuts import render, HttpResponse, redirect**  **….**  **….**  **….**  **def rango2(request,a=0,b=100):**  **if a>b:**  **return redirect('rango2',a=b, b=a)**  **resultado = f"""**  **<h2> Números de [{a},{b}] </h2>**  **…..**  **…..** |
| --- |

**Al cargar**

****

**Django**

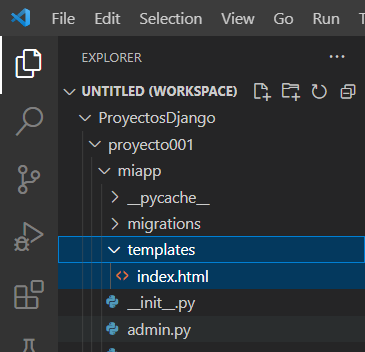
1. **PLANTILLAS Y TEMPLATES**
   1. **Introducción**

Considerar:

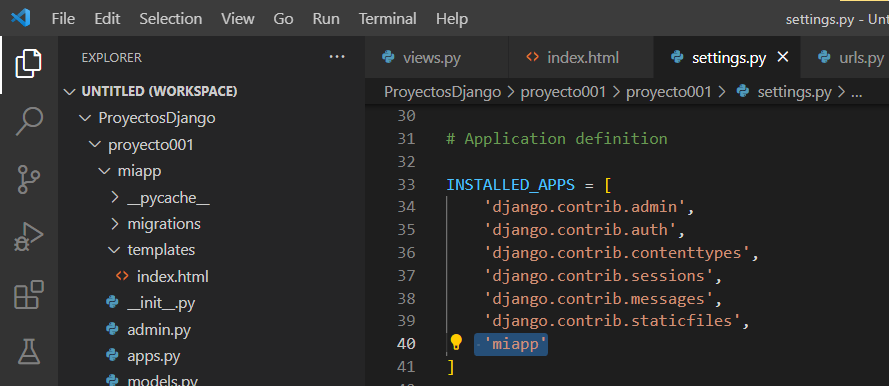
**MVC**: Modelo Vista Control

**MVT**: Modelo Template Vista

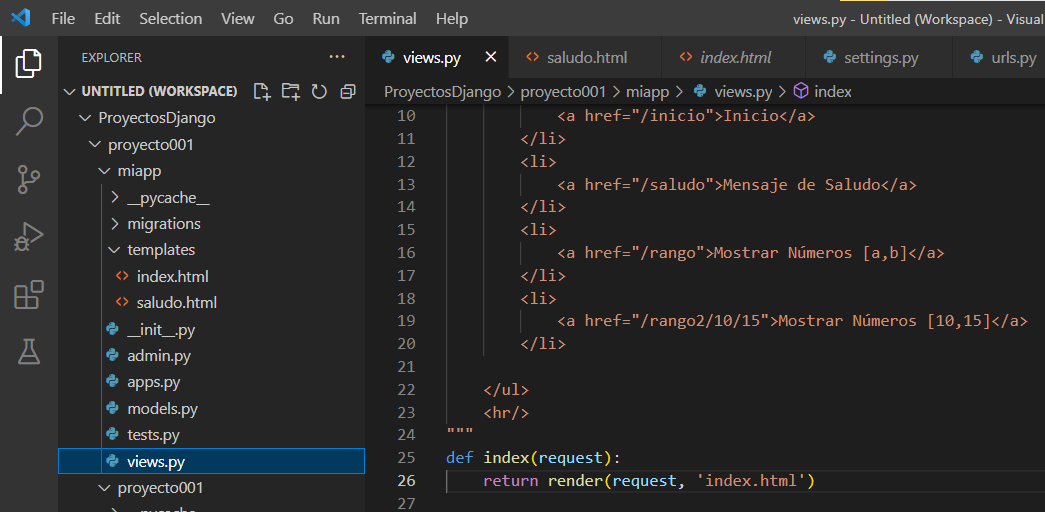
* 1. **En miapps, creamos una carpeta llamada templates y en ella creamos un archivo llamado: index.html**

****

* 1. **En Proyecto001/settings.py agregamos en INSTALLED\_APP: ‘miapp’**

****

* 1. **En Views.py, en el método index, el contenido de mensaje, lo copiamos a templates/index.html**
  2. **En views.py, el método index quedaría:**

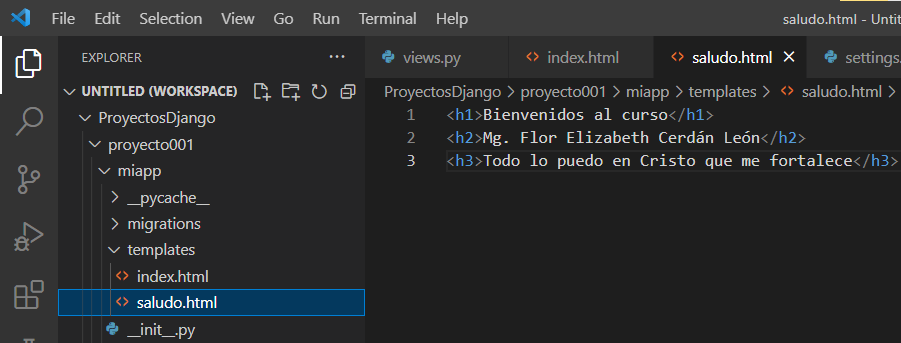
****

**Al cargar**

****

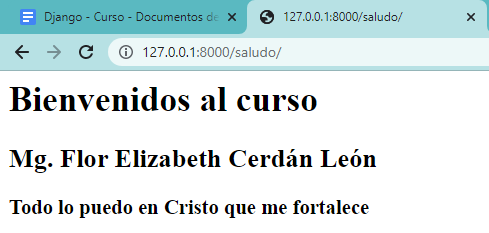
* 1. **Entonces para el saludo hacemos lo mismo:**
  2. **En template, creamos: saludo.html**
  3. **El contenido de mensaje en el método saludo lo cortamos y pegamos en saludo.html**
  4. **En el método saludo configuramos el return**

**En saludo.html**

****

**En views.py**

| **def saludo(request):**  **return render(request,'saludo.html')** |
| --- |

****

**Django**

**Layout, Herencia y Bloques**

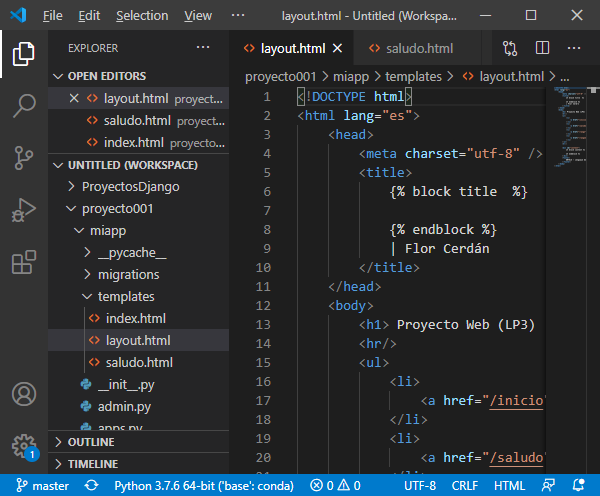
1. Introducción

Considerar:

**MVC**: Modelo Vista Control

**MVT**: Modelo Template Vista

1. En templates creamos el archivo **layout.html**

****

1. En **layout.html** vamos construyendo la plantilla que nuestras páginas van a heredar

| <!DOCTYPE html>  <html lang="es">  <head>  <meta charset="utf-8" />  <title>    </title>  </head>  <body>  <div id="content">    </div>  <footer>  </footer>  </body>  </html> |
| --- |

1. En **layout.html**, creamos el contenido del **title**, del **footer** y copiamos el contenido de la varible layout que está en **views.py**

| <!DOCTYPE html>  <html lang="es">  <head>  <meta charset="utf-8" />  <title>  | Flor Cerdán  </title>  </head>  <body>  <h1> Proyecto Web (LP3) | Flor Cerdán </h1>  <hr/>  <ul>  <li>  <a href="/inicio"> Inicio</a>  </li>  <li>  <a href="/saludo"> Mensaje de Saludo</a>  </li>  <li>  <a href="/rango"> Mostrar Números [a,b]</a>  </li>  <li>  <a href="/rango2/10/15"> Mostrar Números [10,15]</a>  </li>  </ul>  <hr/>  <div id="content">    </div>  <footer>  UNTELS - Lenguaje de Programación III &copy; Flor Cerdán  </footer>  </body>  </html> |
| --- |

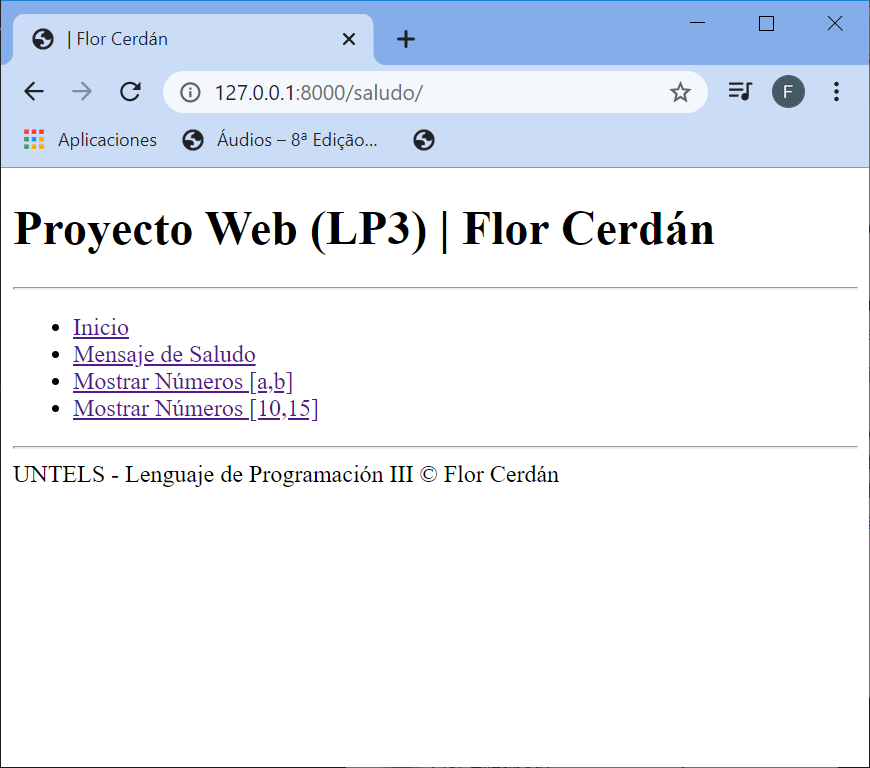
1. En **layout.html**, creamos los bloques para **title** y para **content**.

| **<!DOCTYPE html>**  **<html lang="es">**  **<head>**  **<meta charset="utf-8" />**  **<title>**  **{% block title %}**  **{% endblock %}**  **| Flor Cerdán**  **</title>**  **</head>**  **<body>**  **<h1> Proyecto Web (LP3) | Flor Cerdán </h1>**  **<hr/>**  **<ul>**  **<li>**  **<a href="/inicio"> Inicio</a>**  **</li>**  **<li>**  **<a href="/saludo"> Mensaje de Saludo</a>**  **</li>**  **<li>**  **<a href="/rango"> Mostrar Números [a,b]</a>**  **</li>**  **<li>**  **<a href="/rango2/10/15"> Mostrar Números [10,15]</a>**  **</li>**  **</ul>**  **<hr/>**  **<div id="content">**  **{% block content %}**  **{% endblock %}**  **</div>**  **<footer>**  **UNTELS - Lenguaje de Programación III &copy; Flor Cerdán**  **</footer>**  **</body>**  **</html>** |
| --- |

1. En **saludo.html**, utilizamos el comando **extends** para que herede el contenido de **layout.html.**

| **{% extends 'layout.html' %}**  **<h1>Bienvenidos</h1>**  **<h2>Flor Cerdán</h2>**  **<h3>Python....</h3>** |
| --- |

Al cargar la página, se observa que en **saludo/**, está cargando el contenido de **layout.html**, sin embargo no se está personalizando los bloques: **title** y **content**.

****

1. En **saludo.html**, definimos los bloques **content** y **title**.

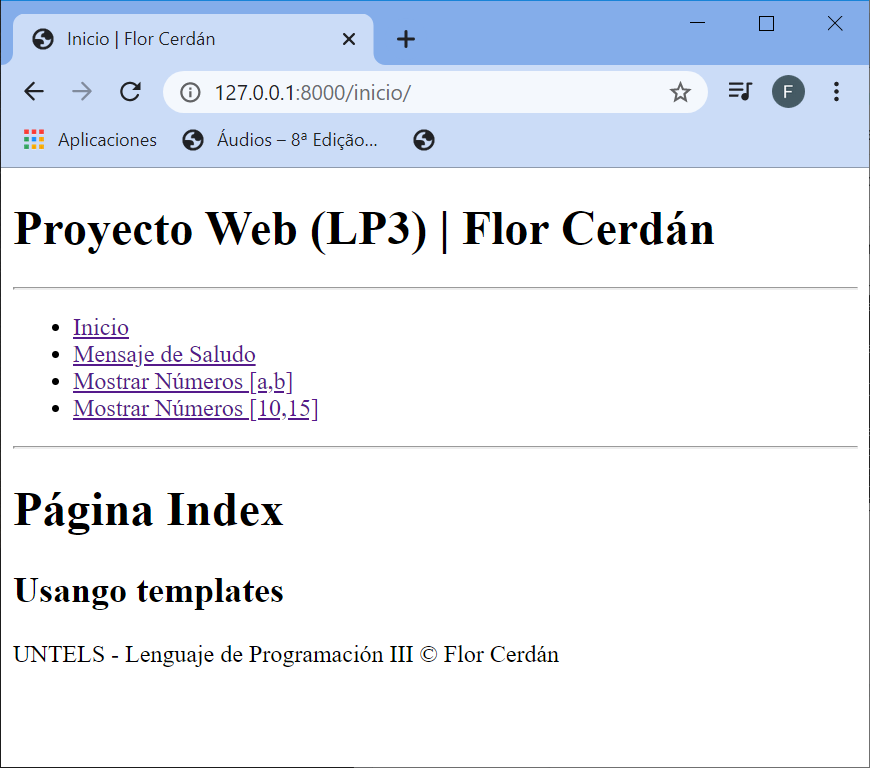
| {% extends 'layout.html' %}  {% block title %}  Saludo  {% endblock %}  {% block content %}  <h1>Bienvenidos</h1>  <h2>Flor Cerdán</h2>  <h3>Python....</h3>  {% endblock %} |
| --- |

Ahora, al cargar la página **saludo/** se observa que el mensaje del **title** y **content** ha cambiado según lo definido en **saludo.html**



1. Finalmente hacemos lo mismo en **index.html**: Definimos los bloques **content** y **title**.

| {% extends 'layout.html' %}  {% block title %}  Inicio  {% endblock %}  {% block content %}  <h1>Página Index</h1>  <h2>Usango templates</h2>  {% endblock %} |
| --- |



1. Dentro del **block content** en **layout.html**, ponemos un mensaje (Contenido por defecto); y en **saludo.html**, eliminamos el **block content**

**layout.html**

| ...  </ul>  <hr/>  <div id="content">  {% block content %}  <strong> Contenido en layout (Padre) </strong>  {% endblock %}  </div>  <footer>  UNTELS - Lenguaje de Programación III &copy; Flor Cerdán  </footer>  ... |
| --- |

**saludo.html**

| {% extends 'layout.html' %}  {% block title %}  Saludo  {% endblock %} |
| --- |

Al cargar la página saludo/, se observa lo siguiente:

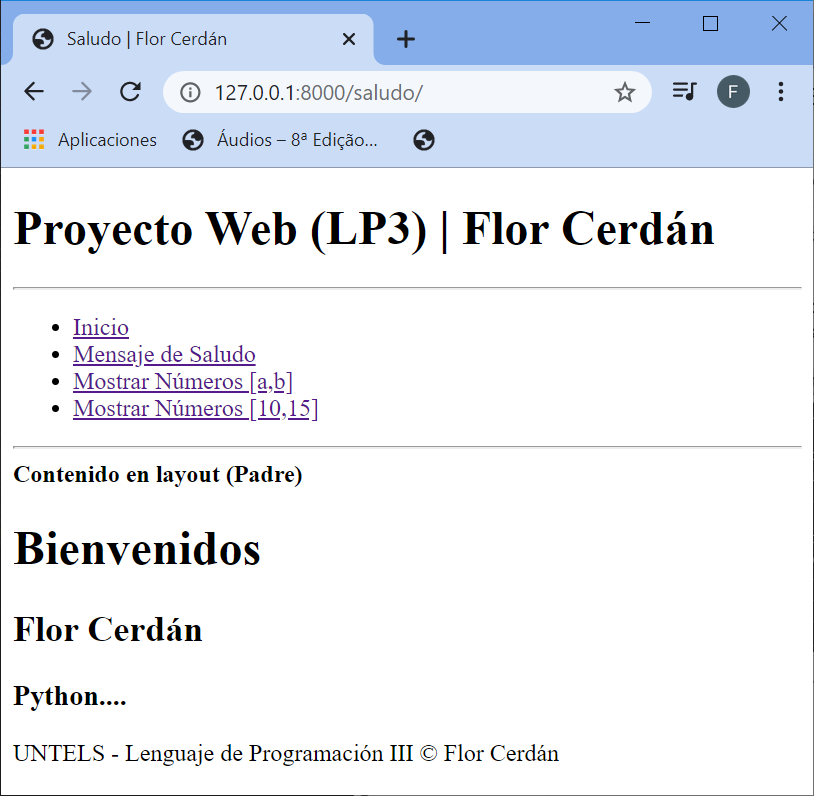


Eso significa, que si no defino un **block content**, en **saludo.html**, se mostrará el contenido del **block content** de **layout.html**.

1. Finalmente, si deseo mostrar el **block content** de saludo.html y el contenido del **block content** de **layout.html**, en **saludo.html** invocamos el **block content** del padre utilizando **block.super**.

**saludo.html**

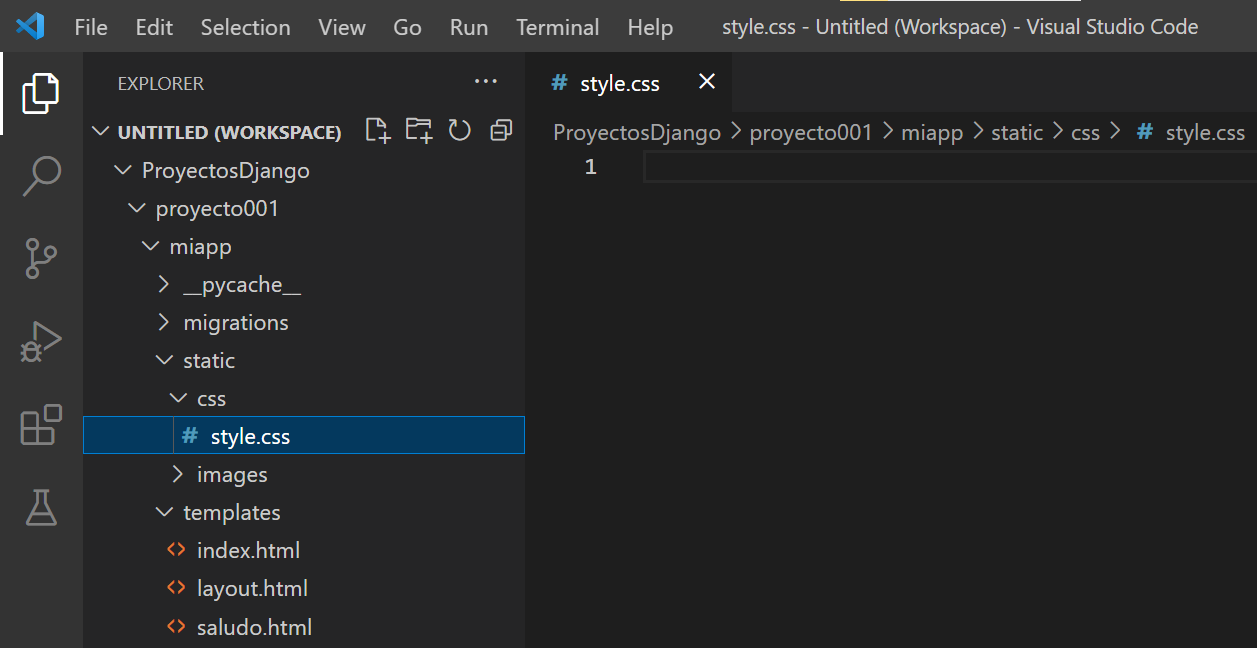
| {% extends 'layout.html' %}  {% block title %}  Saludo  {% endblock %}  {% block content %}  {{ block.super }}  <h1>Bienvenidos</h1>  <h2>Flor Cerdán</h2>  <h3>Python....</h3>  {% endblock %} |
| --- |



**Django**

**Estilos CSS y Maquetación**

1. Dentro de **miapp**: **static** y dentro de **static** las carpetas: **css** e **images**, finalmente dentro de **css** creamos el archivo llamado **style.css**

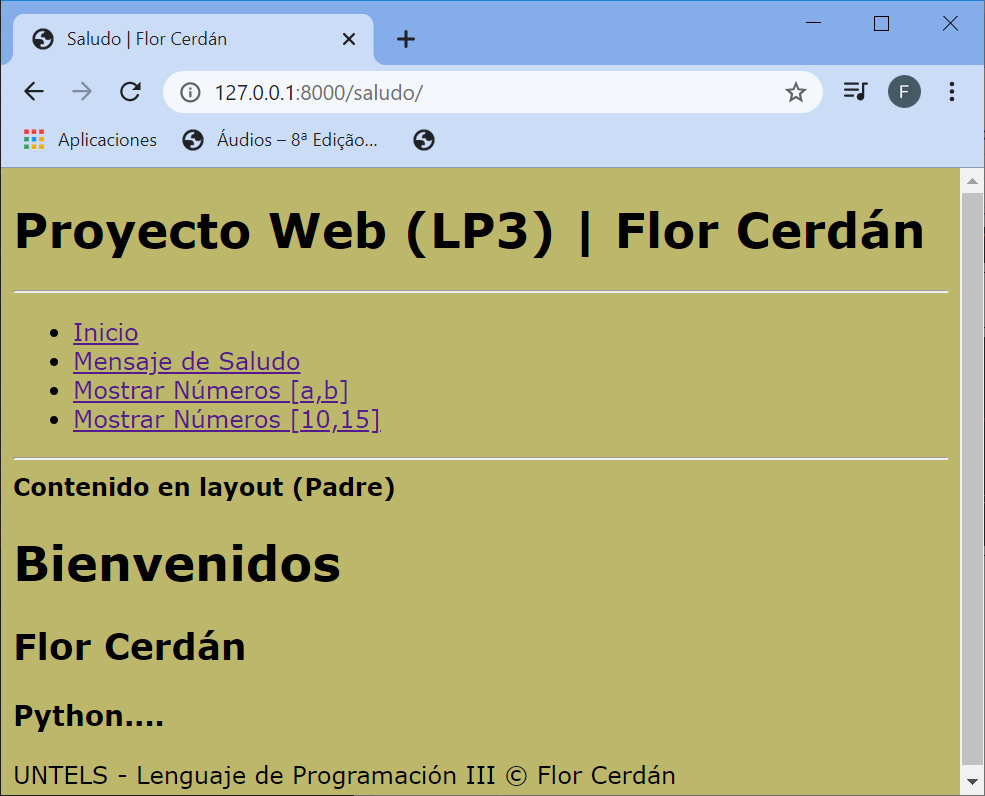
****

1. En **style.css** definimos un estilo para el color de fondo:

| body{  font-family: Verdana, Geneva, Tahoma, sans-serif;  background-color: darkkhaki;  } |
| --- |

1. En layout.html, vinculamos el estilo.

| ...  | Flor Cerdán  </title>  {% load static %}  <link rel="stylesheet" type="text/css" href="{% static 'css/style.css' %}" />  </head>  <body>  <h1> Proyecto Web (LP3) | Flor Cerdán </h1>  <hr/>  ... |
| --- |

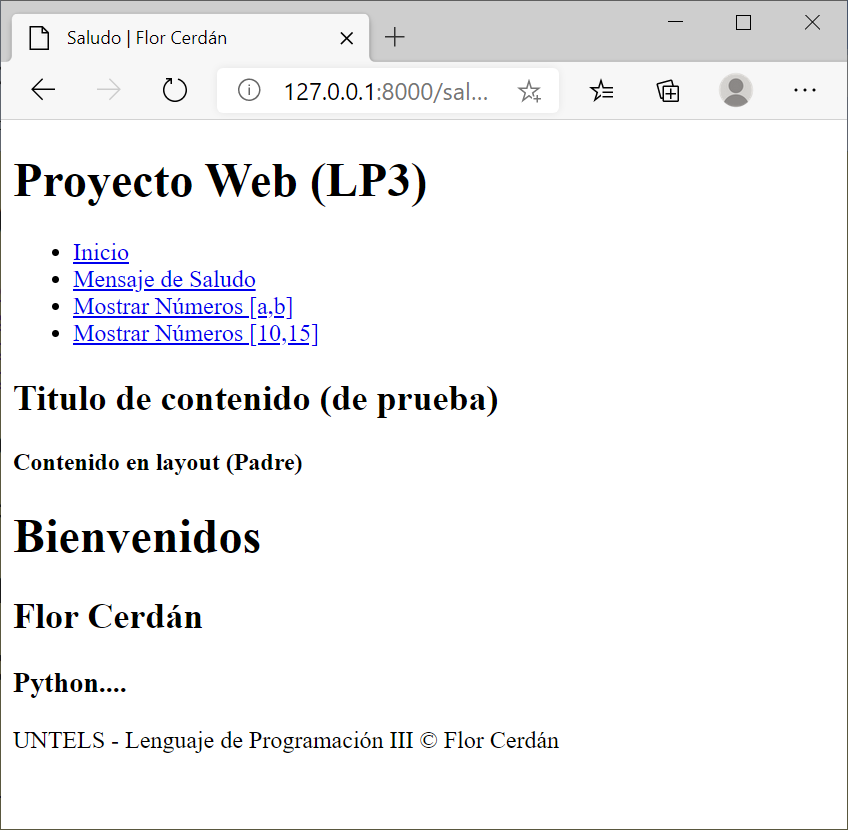


**Django**

**Estilos CSS y Maquetación**

1. Eliminamos el contenido del archivo: **styles.css**
2. Ahora vamos a mejorar la estructura de nuestra plantilla: **layout.html**. Lo vamos a estructurar con un **encabezado**, un **menú** y una **parte de contenido** con un estilo más agradable.

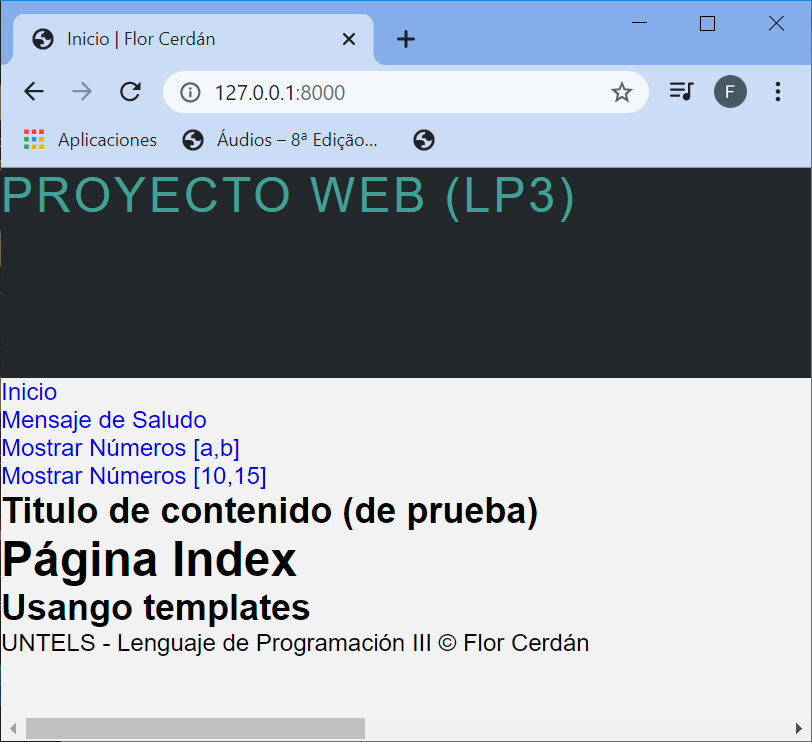
| ...  <body>  **<header>**  **<div id="logotipo">**  **<h1>Proyecto Web (LP3)</h1>**  **</div>**  **</header>**  **<nav>**  <ul>  <li>  <a href="/inicio"> Inicio</a>  </li>  <li>  <a href="/saludo"> Mensaje de Saludo</a>  </li>  <li>  <a href="/rango"> Mostrar Números [a,b]</a>  </li>  <li>  <a href="/rango2/10/15"> Mostrar Números [10,15]</a>  </li>  </ul>  **</nav>**  <div id="content">  **<h2 class="title">Titulo de contenido (de prueba)</h2>**  {% block content %}  <strong> Contenido en layout (Padre) </strong>  {% endblock %}  </div>  <footer>  UNTELS - Lenguaje de Programación III &copy; Flor Cerdán  </footer>  </body>  ... |
| --- |



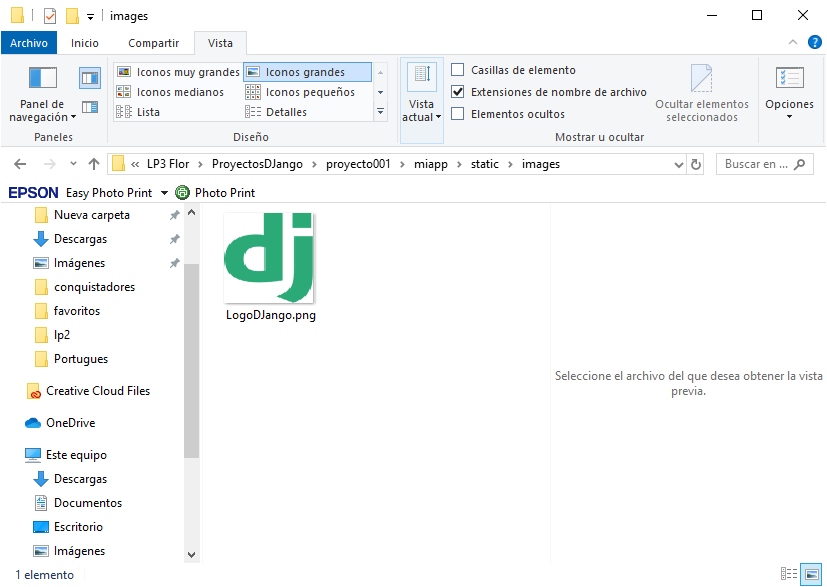
**ESTILOS AL HEADER**

1. En **Style.css** creamos: **Estilos Generales** y **Estilos para la Cabecera**.

| /\*Estilos Generales\*/  \*{  margin: 0px;  padding: 0px;  font-family: Arial, Helvetica, sans-serif;  text-decoration: none;  }  body{  background-color: #f2f2f2;  }  /\* Estilos Cabecera \*/  header{  width: 1212px;  height: 140px;  background-color: #23282B;  margin: 0 auto;  }  #logotipo h1{  display: block;  float: left;  margin-top: 35px;  margin-left: 20px;  letter-spacing: 2px;  font-weight: lighter;  color: #2ba977;  } |
| --- |



1. En la carpeta images, vamos a copiar el archivo LogoDJango.png



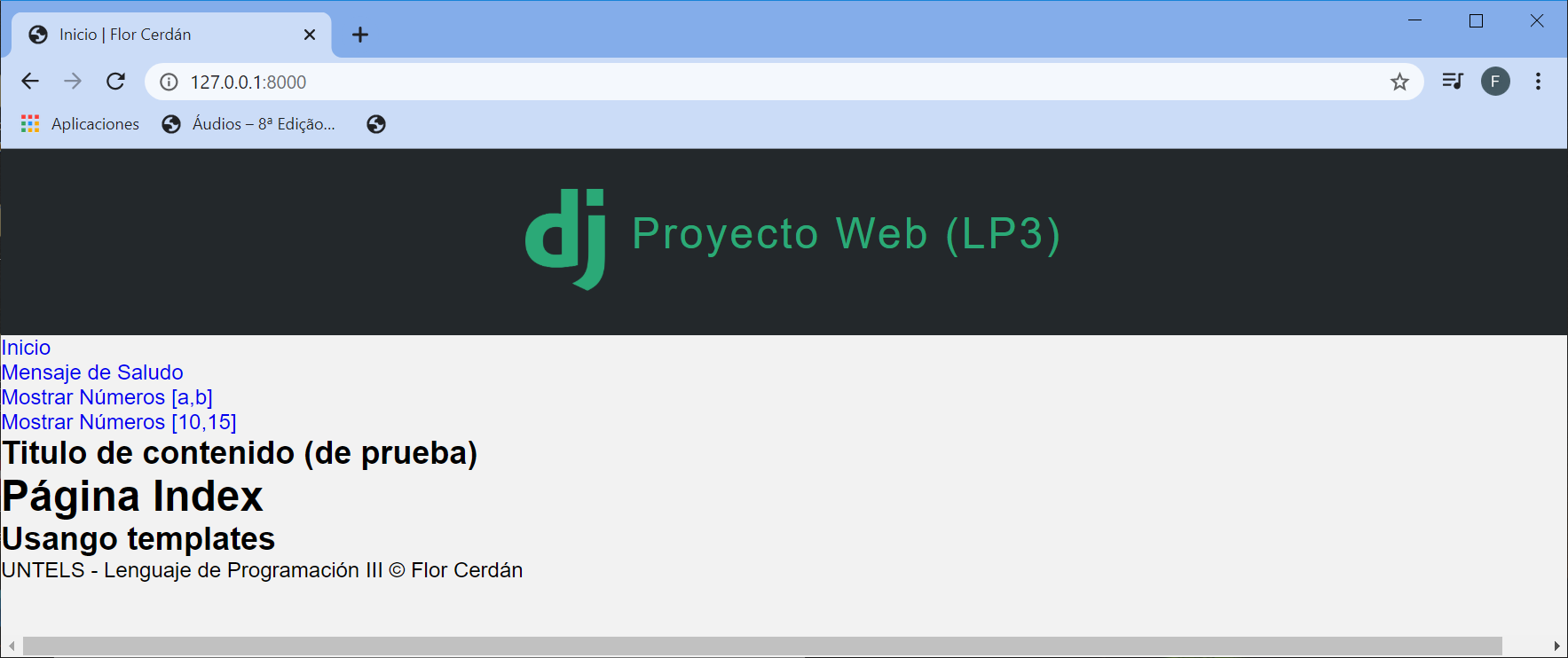
1. Para cargar el logo dentro del header:

Layout.html

| ...  <header>  <div id="logotipo">  **<img src="{% static 'images/LogoDJango.png' %}" />**  <h1>Proyecto Web (LP3)</h1>  </div>  </header>  ... |
| --- |

Style.css

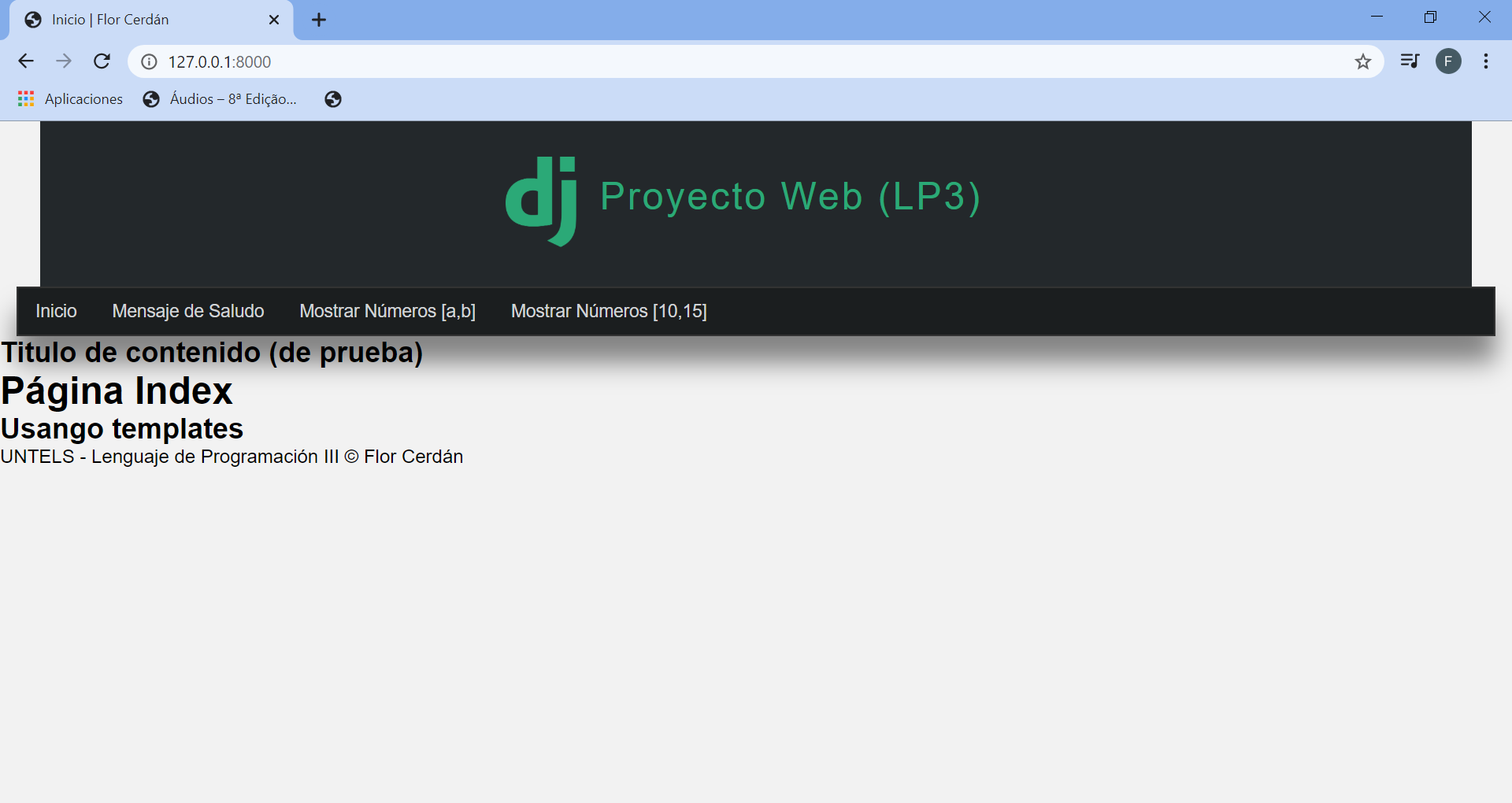
| /\*Estilos Generales\*/  \*{  margin: 0px;  padding: 0px;  font-family: Arial, Helvetica, sans-serif;  text-decoration: none;  }  body{  background-color: #f2f2f2;  }  /\* Estilos Cabecera \*/  header{  width: 1212px;  height: 140px;  background-color: #23282B;  margin: 0 auto;  }  #logotipo {  width: 35%;  height: 130px;  margin: 0 auto;  padding-top: 10px;  }  #logotipo img{  display: block;  width: 60px;  float: left;  margin-top: 20px;  }  #logotipo h1{  display: block;  float: left;  margin-top: 35px;  margin-left: 20px;  letter-spacing: 2px;  font-weight: lighter;  color: #2ba977;  } |
| --- |



**ESTILOS A LA BARRA DE NAVEGACIÓN**

1. Agregamos en Style.css, el estilo para la zona de navegación

| /\*Estilos Navegación\*/  /\*Zona de menu\*/  nav{  width: 1250px;  height: 40px;  margin: 0 auto;  background-color: #1B1E1F;  border: 1px solid #333333;  box-shadow: 0px 22px 22px gray;  font-size: 15px;  }  /\*Zona de enlaces\*/  nav ul{  list-style: none;  text-decoration: none;  }  nav ul li{  line-height: 40px;  float: left;  }  /\*Hipervinculos\*/  nav ul li a{  display: block;  padding-left: 15px;  padding-right: 15px;  color: #d1d4d6;  }  /\*Efecto en Hipervínculos\*/  nav ul li a:hover{  background: #2ba977;  box-shadow: 0px 0px 0px #444 inset;  color:white;  transition: all 300ms;  } |
| --- |



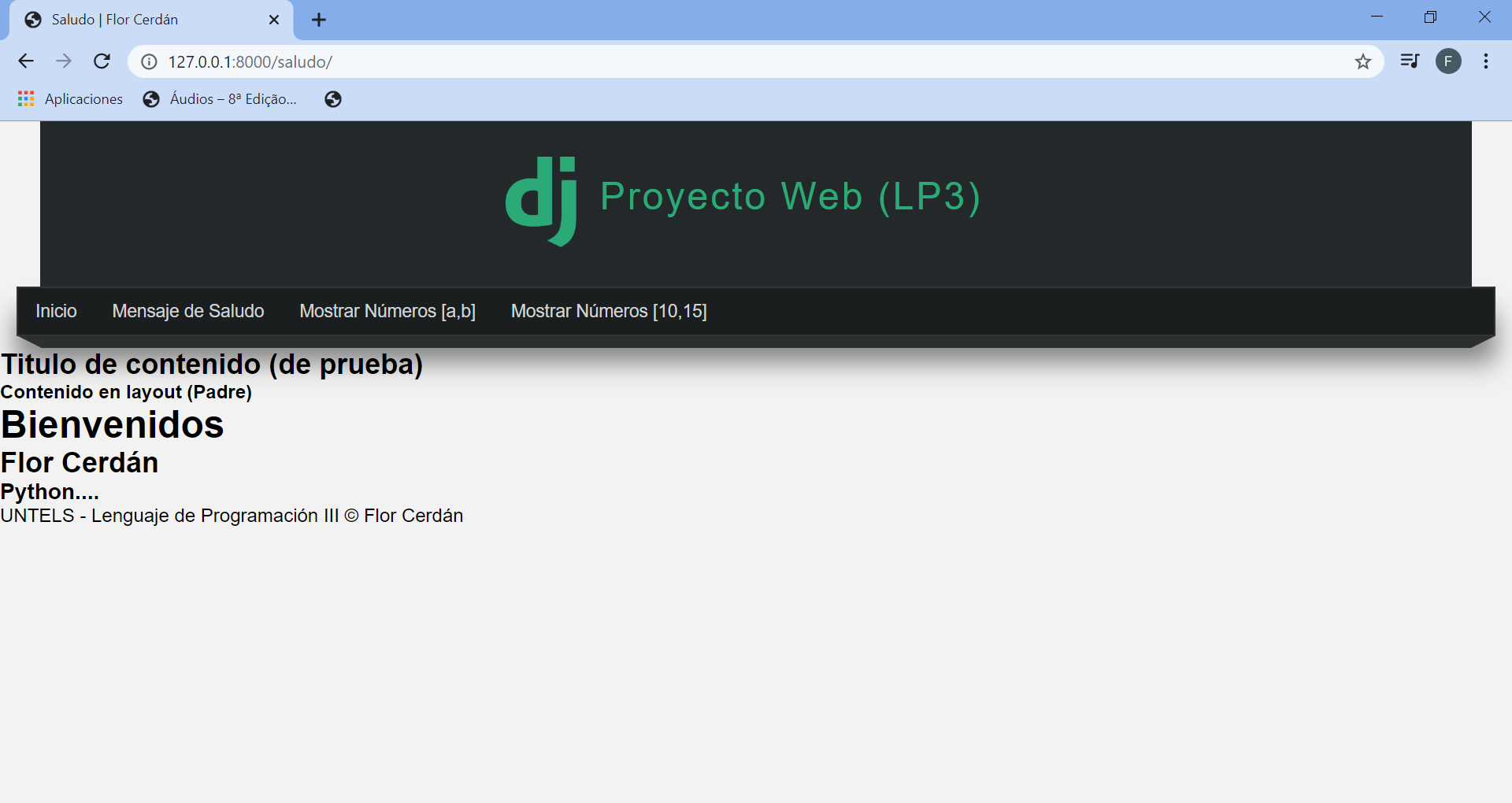
1. Vamos a diseñar un efecto 3D al menú de opciones

**Layout.html**

| ...  </li>  </ul>  </nav>  **<div class="barra">**  **<span class="esquina-izquierda"></span>**  **<span class="relleno-barra"></span>**  **<span class="esquina-derecha"></span>**  **</div>**  <div id="content">  ... |
| --- |

**Style.css**

| ...  /\*Efecto 3D\*/  .barra{  width: 1250px;  margin: 0 auto;  }  .esquina-izquierda{  display: block;  border-left: 20px solid transparent;  border-top: 10px solid #2B2F30;  float: left;  }  .esquina-derecha{  display: block;  border-right: 20px solid transparent;  border-top: 10px solid #2B2F30;  float: right;  }  /\*Relleno del hueco\*/  .relleno-barra{  display: block;  width: 1210px;  height: 10px;  background: #2B2F30;  float: left;  } |
| --- |



**ESTILOS AL CONTENIDO**

1. Para crear un cuadro dentro del contenido:

**Layout.html**

| ...  <div id="content">  **<div class="box">**  <h2 class="title">Titulo de contenido (de prueba)</h2>  {% block content %}  <strong> Contenido en layout (Padre) </strong>  {% endblock %}  **</div>**  </div>  ... |
| --- |

**Style.css**

| /\*Estilos para el CONTENIDO\*/  #content{  width: 1212px;  min-height: 930px;  margin: 0 auto;  margin-bottom: 30px;  margin-top: 40px;  }  .box{  background: white;  width: 95%;  min-height: 930px;  padding: 20px;  border: 1px solid #ddd;  border-radius: 2px;  margin: 0 auto;  } |
| --- |



1. Para agregar estilo al texto del contenido

* En Layout.html, eliminamos:

<h2 class="title">Titulo de contenido (de prueba)</h2>

* En Style.css: creamos: .title{ }
* En index.html y saludo.html, configuramos: class=”title”

**Layout.html**

| ...  <div id="content">  **<div class="box">**    {% block content %}  <strong> Contenido en layout (Padre) </strong>  {% endblock %}  **</div>**  </div>  ... |
| --- |

**Style.css**

| ...  .title{  color: #444;  letter-spacing: 1px;  font-size: 30px;  margin-bottom: 4px;  margin-top: 5px;  } |
| --- |

**index.html**

| {% extends 'layout.html' %}  {% block title %}  Inicio  {% endblock %}  {% block content %}  <h1 **class="title"**>Página Index</h1>  <h2>Usango templates</h2>  {% endblock %} |
| --- |

**saludo.html**

| {% extends 'layout.html' %}  {% block title %}  Saludo  {% endblock %}  {% block content %}  <h1 **class="title"**>Bienvenidos</h1>  <h2>Flor Cerdán</h2>  <h3>Python....</h3>  **{{ block.super }}**  {% endblock %} |
| --- |

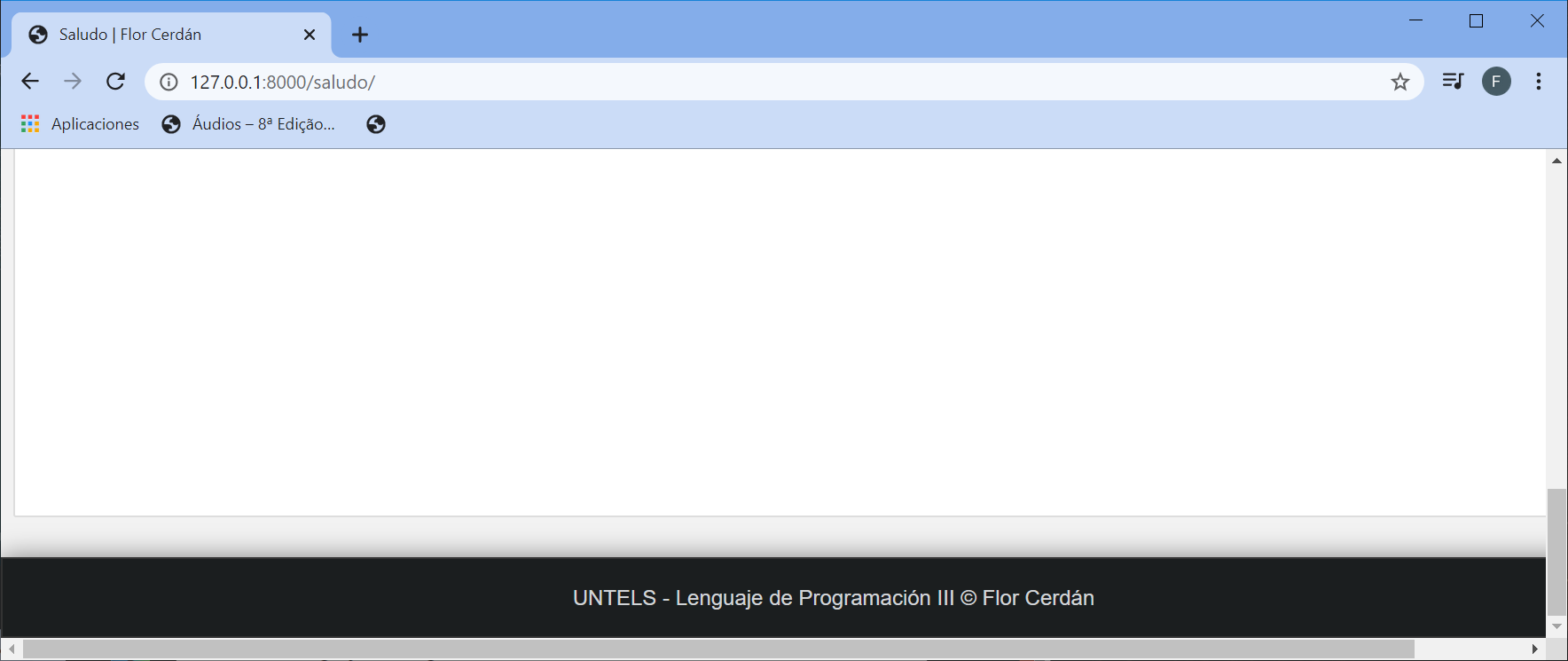


ESTILO PARA FOOTER

1. Para agregar estilo al texto del footer:

**Style.css**

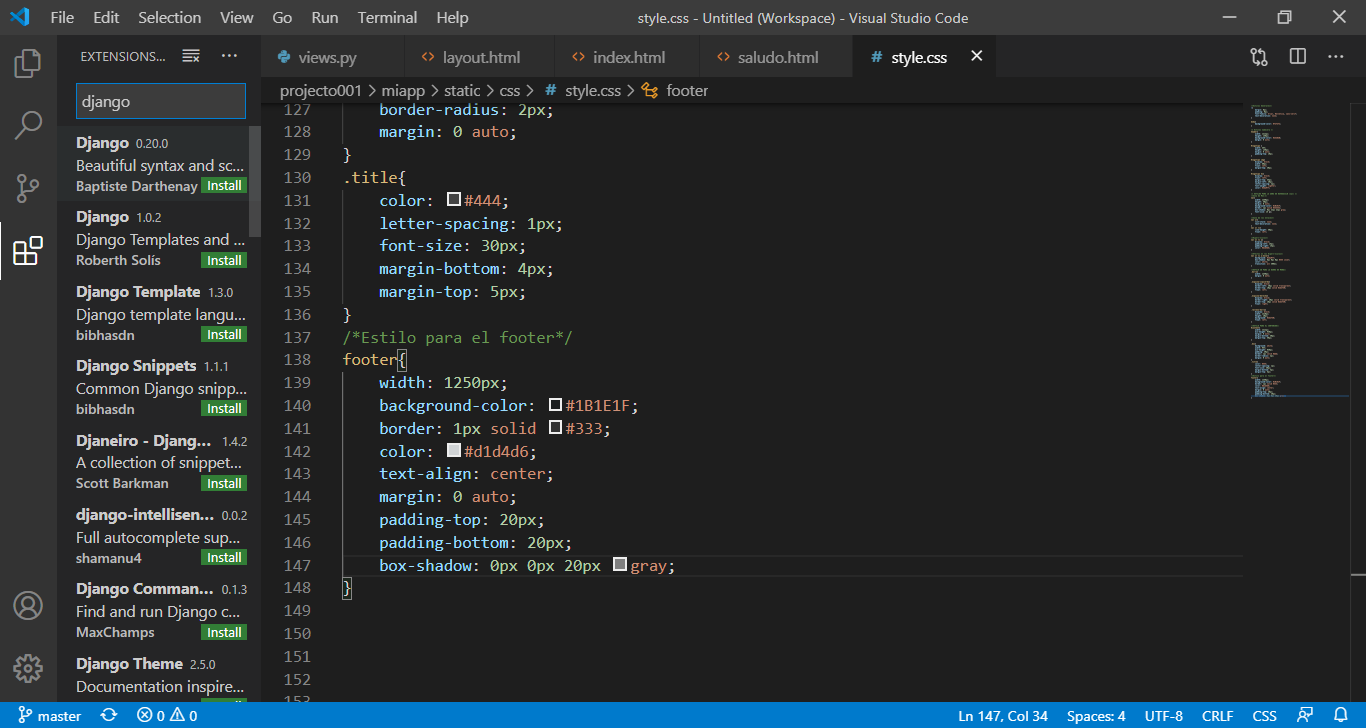
| /\*Estilo para el footer\*/  footer{  width: 1250px;  background-color: #1B1E1F;  border: 1px solid #333;  color: #d1d4d6;  text-align: center;  margin: 0 auto;  padding-top: 20px;  padding-bottom: 20px;  box-shadow: 0px 0px 20px gray;  } |
| --- |

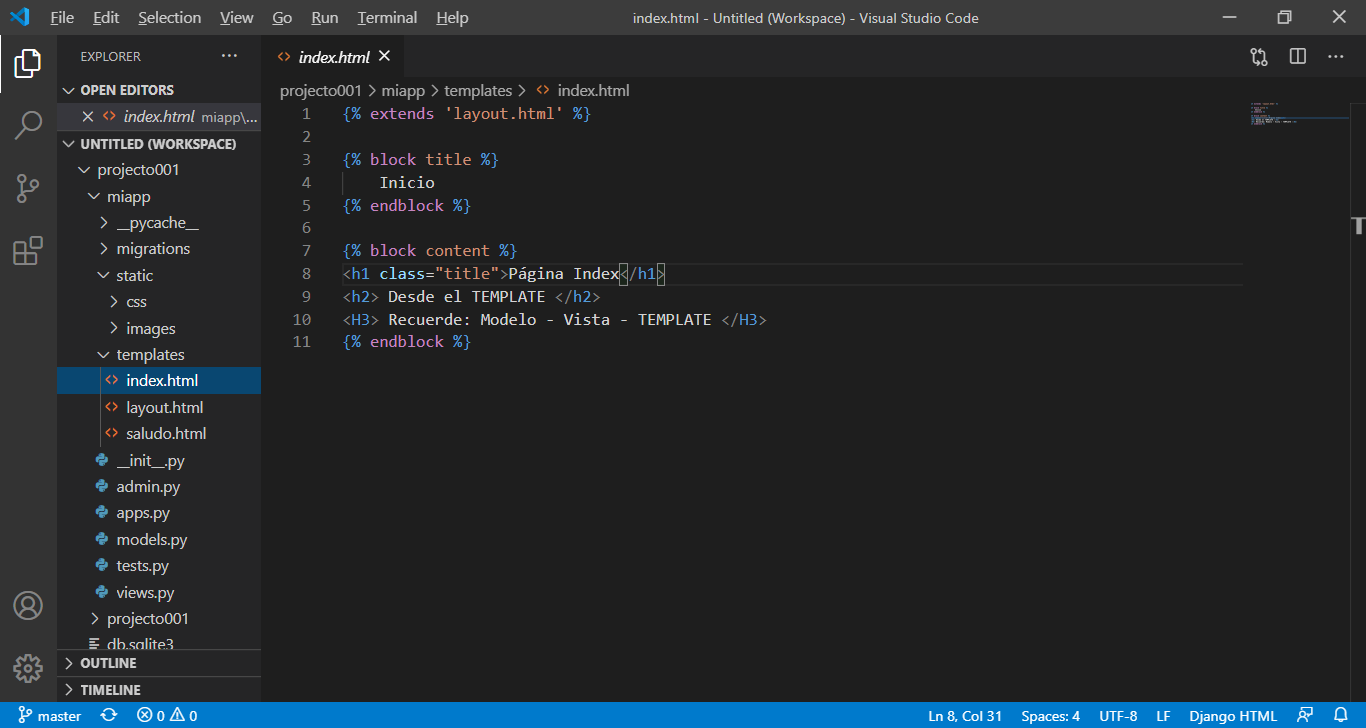


**Django**

**Lenguaje de Plantillas y Template Tags**

1. Instalar una extensión para resaltar los bloque de código en Django para Visual Studio Code





1. Comentarios en html y en bloques DJango

En html utilizamos: <!-- …. -->

En DJango utilizamos: {% comment ‘ Título del comentario ’ %} …. {%endcomment%}

* En el caso de los comentarios en DJango, este cuenta con una estructura, es decir, tiene un título del comentario y un cuerpo del comentario

**index.html**

| {% extends 'layout.html' %}  {% block title %}  Inicio  {% endblock %}  {% block content %}  <!--Comentario en HTML-->  {% comment 'Datos de la pagina index' %}  En esta pagina se mostrara la bienvenida al sistema  {% endcomment %}  <h1 class="title">Página Index</h1>  <h2>Usango templates</h2>  {% endblock %} |
| --- |

1. Pasar datos de la vista a la plantilla

En **view.py**

| def index(request):  return render(request,'index.html', ***{***  ***'mensaje':'Proyecto web con DJango (Desde el View)'***  ***}***) |
| --- |

En **templates/index.html**

| ...  <h1 class="title">Página Index</h1>  <h2>Usango templates</h2>  **Mensaje: {{mensaje}}**  {% endblock %} |
| --- |



1. Ahora, podemos también enviar **titulo = ‘Inicio’** del **view.py → Index.html**

En **view.py**

| def index(request):  return render(request,'index.html', {  'titulo':'Inicio',  'mensaje':'Proyecto web con DJango (Desde el View)'  }) |
| --- |

En **templates/index.html**

| ...  {% block title %}  **{{titulo}}**  {% endblock %}  ...  ...  <h1 class="title">Página **{{titulo}}**</h1>  <h2>Usango templates</h2>  ... |
| --- |



1. Ejercicio:
   1. Para **saludo.html**:

* Enviar el **título** desde el **view**
* Enviar **nombre\_autor** desde el view

**SOLUCIÓN:**

En **view.py**

| def saludo(request):  return render(request,'saludo.html'**,{**  **'titulo':'Saludo',**  **'autor\_saludo':'Mg. Flor Elizabeth Cerdán León'**  **}**) |
| --- |

En **templates/saludo.html**

| ...  {% block title %}  **{{titulo}}**  {% endblock %}  {% block content %}  <h1 class="title">Bienvenidos</h1>  <h2>**{{autor\_saludo}}**</h2>  ... |
| --- |



**ESTRUCTURAS CONDICIONALES**

1. En **index.html**, si el mensaje no tiene contenido mostrar: **“El mensaje no tiene contenido”**

En **style.css**

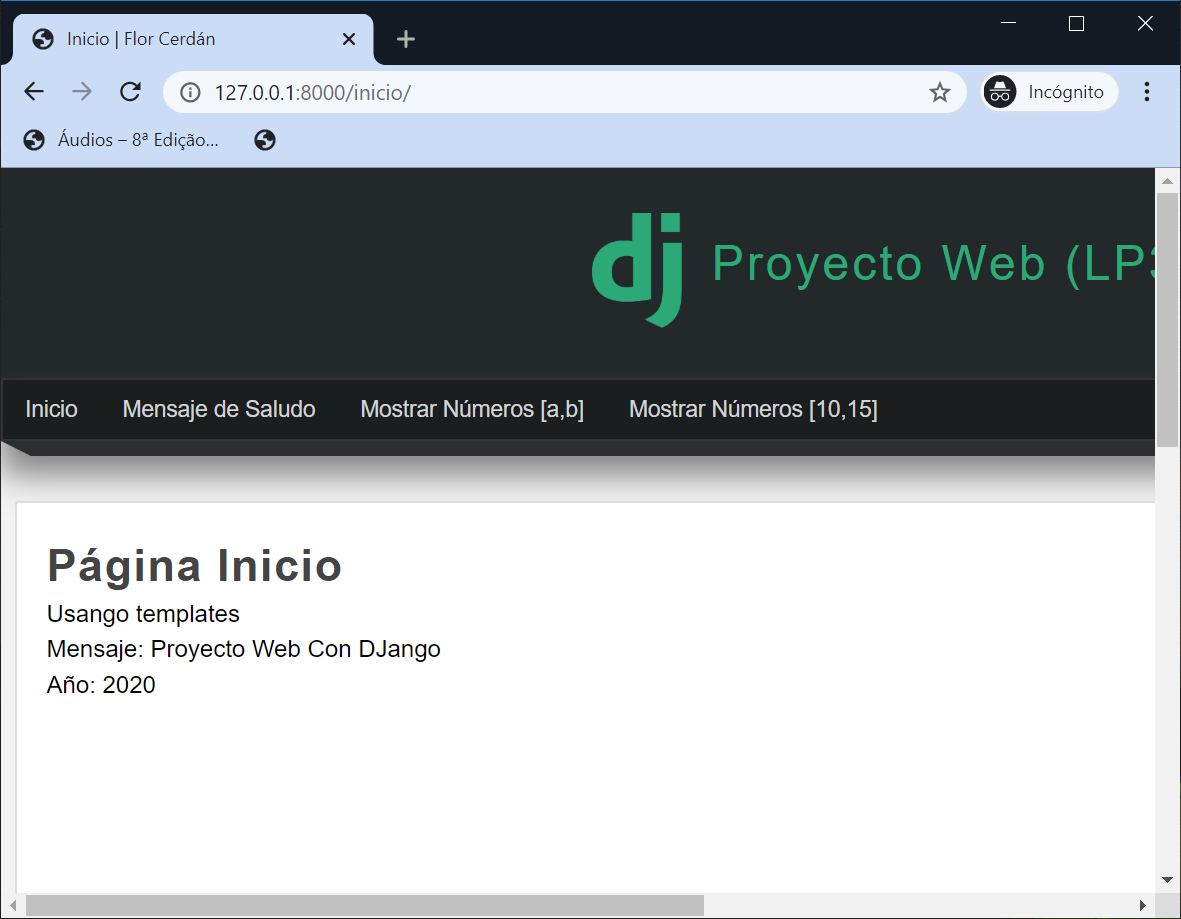
| ...  .box p, .box h2, .box h3, .box ul{  margin-bottom: 5px;  margin-top: 5px;  }  .box ul, .box ol{  margin-left: 20px;  margin-bottom: 10px;  }  body{  ... |
| --- |

En **view.py**

| ...  def index(request):  return render(request,'index.html', {  'titulo':'Inicio',  'mensaje':'Proyecto Web Con DJango'  })  ... |
| --- |

En **templates/index.html**

| ...  <h1 class="title">Página {{titulo}}</h1>  <p>Usango templates</p>  {% if mensaje %}  <p> Mensaje: {{mensaje}} </p>  {% else %}  <strong>El Mensaje no tiene contenido</strong>  {% endif %}    <p>Año: 2020</p>  ... |
| --- |



**BUCLES**

1. Enviar una lista del **view.py** a **index.html**, utilizando **la estructura repetitiva “for”**, mostrar su contenido.

En **view.py**

| ...  def index(request):  estudiantes = [ 'Isabella Caballero',  'Alejandro Hermitaño',  'Joan Palomino',  'Pierre Bernaola']  return render(request,'index.html', {  'titulo':'Inicio',  'mensaje':'Proyecto Web Con DJango',  'estudiantes': estudiantes  })  ... |
| --- |

En **templates/index.html**

| ...  {% if mensaje %}  <p> Mensaje: {{mensaje}} </p>  {% else %}  <strong>El Mensaje no tiene contenido</strong>  {% endif %}  <p> <strong> Estudiantes Registrados (Método 01: Imprime lista): </strong> </p>  {{estudiantes}}  <p> <strong> Estudiantes Registrados (Método 02: Imprime lista utilizando for): </strong> </p>  <ul>  {% for estudiante in estudiantes %}  <li>{{forloop.counter}}. {{estudiante}}</li>  {% endfor %}  </ul>  <p>Año: 2022</p>  ... |
| --- |





1. ¿Qué pasaría si la lista está vacía?

En **templates/index.html**

| ...  {% for estudiante in estudiantes %}  <li>{{forloop.counter}}. {{estudiante}}</li>  {% empty %}  <p> No hay estudiantes registrados </p>  {% endfor %}  ... |
| --- |

En **view.py**

| ...  def index(request):  estudiantes = ['Patricia Espinola',  'Deyli Diaz',  'Milton Medrano',  'Walter Vargas']  estudiantes = []  ... |
| --- |



1. **Ejercicio**:

En base a todo lo aprendido.

* En view.py
  + Calcular el rango en una lista: rango\_numeros
  + Enviar como parámetro del render:
    - titulo
    - a
    - b
    - rango\_numeros
* Crear la página **rango.html**
  + Heredar **layout.html**
  + Crear **block title**
  + Crear **block content**
  + Mostrar la lista **rango\_numeros**



**Solución:**

En **templates/rango.html**

| {% extends 'layout.html' %}  {% block title %}  {{titulo}}  {% endblock %}  {% block content %}  <h1 class="title">Números de [{{a}} , {{b}}]</h1>  <ul>  {% for numero in rango\_numeros %}  <li>{{numero}} </li>  {% empty %}  Rango vacío  {% endfor %}  </ul>  {% endblock %} |
| --- |

En **view.py**

| ...  def rango(request):  a = 10  b = 20  rango\_numeros = range(a,b+1)  return render(request,'rango.html',{  'titulo':'Rango',  'a':a,  'b':b,  'rango\_numeros':rango\_numeros  })  ... |
| --- |

****

1. **Includes** template con django.

Crear un template: **listado\_estudiantesregistrados.html**, y sea invocado desde **index.html**

En **listado\_estudiantesregistrados.html**

| <br>  <hr>  <br>  <p> <strong> {{encabezado}} (Método 01: Imprime lista): </strong> </p>  {{estudiantes}}  <p> <strong> {{encabezado}} (Método 02: Imprime lista utilizando for): </strong> </p>  <ul>  {% for estudiante in estudiantes %}  <li>{{forloop.counter}}. {{estudiante}}</li>  {% empty %}  <p> No hay {{encabezado}} </p>  {% endfor %}  </ul> |
| --- |

En **index.html**

**Borramos**

| **<p> <strong>Estudiantes Registrados (Método 01: Imprime lista): </strong> </p>**  **{{estudiantes}}**  **<p> <strong>Estudiantes Registrados (Método 02: Imprime lista utilizando for): </strong> </p>**  **<ul>**  **{% for estudiante in estudiantes %}**  **<li> {{forloop.counter}}. {{estudiante}} </li>**  **{% empty %}**  **<p> No hay estudiantes registrados </p>**  **{% endfor %}**  **</ul>**  **<br>** |
| --- |

**Nos debe quedar así:**

| ...  <strong>El Mensaje no tiene contenido</strong>  {% endif %}  {% include 'listado\_estudiantesregistrados.html' with encabezado="Estudiantes Registrados" %}    <p>Año: 2022</p>  {% endblock %} |
| --- |

**Observación: “Si utilizamos only en el include, significa que listado\_estudiantesregistrados.html, solo va a heredar la variable indicada.”**

1. URLs en template

URL En layout.html, vamos a invocar el nombre de los template, según lo que está registrado en **urls.py**.

En **layout.html**

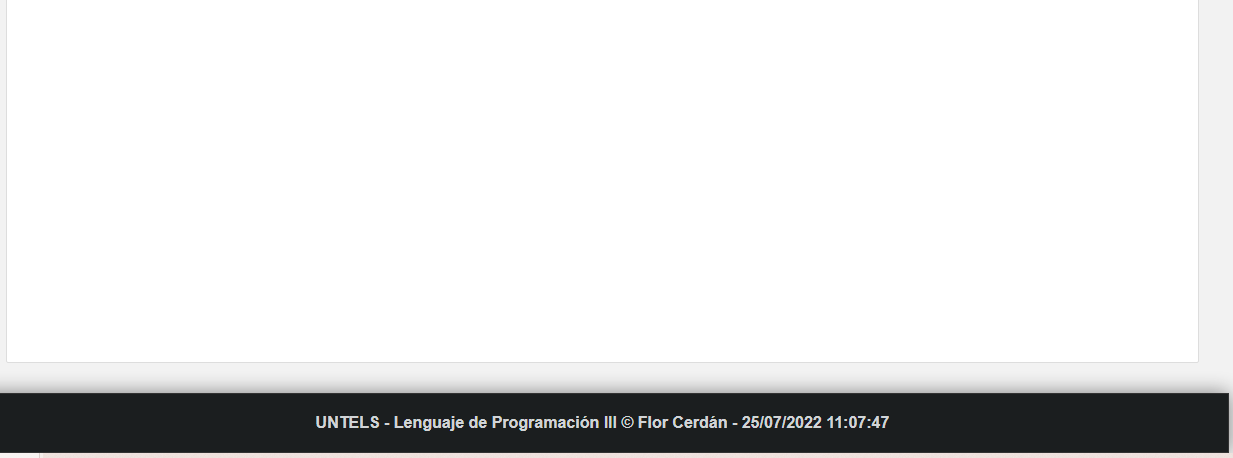
| ...  <li>  <a href="{% url 'inicio' %}"> Inicio</a>  </li>  <li>  <a href="{% url 'saludo' %}"> Mensaje de Saludo</a>  </li>  <li>  <a href="{% url 'rango' %}"> Mostrar Números [a,b]</a>  </li>  <li>  <a href="{% url 'rango2' %}"> Mostrar Números [10,15]</a>  </li>  ... |
| --- |

1. Para obtener la fecha del sistema utilizamos **now**

En **layout.html**

| ...  UNTELS - Lenguaje de Programación III &copy; Flor Cerdán - {% now "d/m/Y h:m:s" %}  ... |
| --- |

Al Cargar



1. Crear un template: **filtros.html** y utilizar los filtros más conocidos para utilizar con cadenas y listas.

En **index.html**

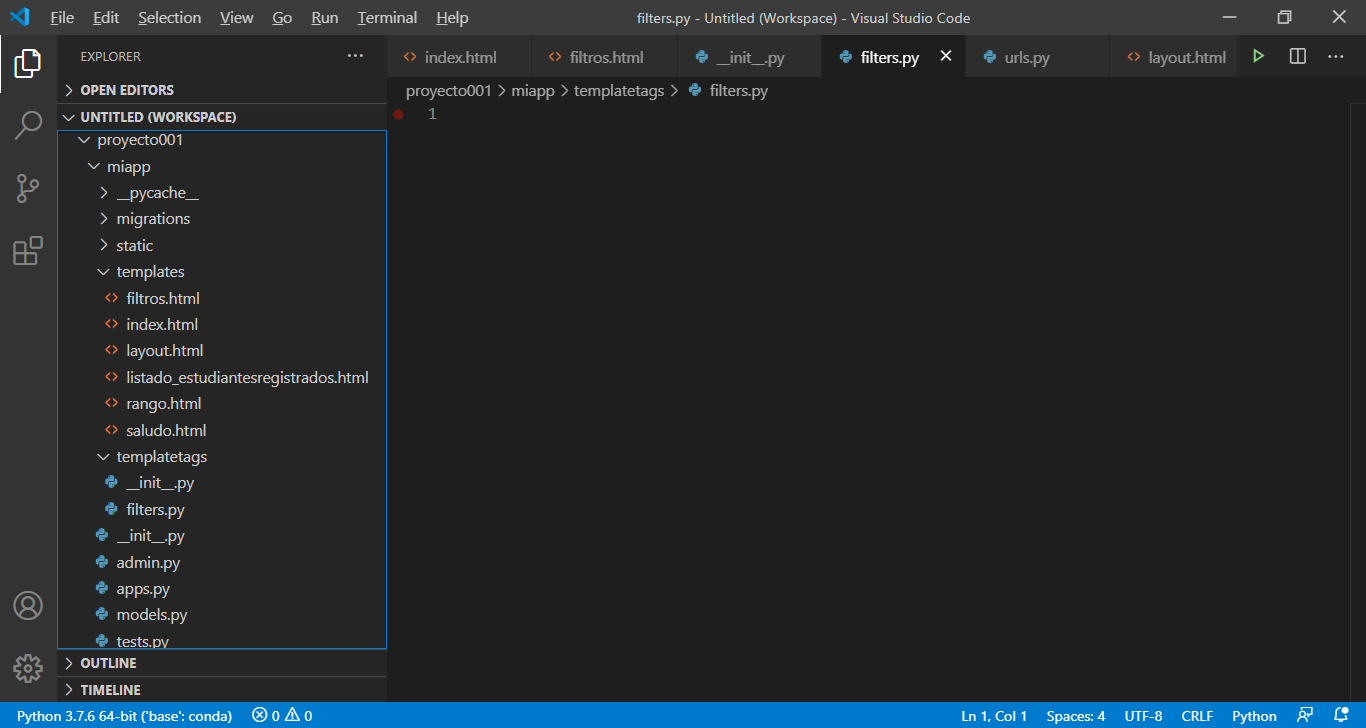
| ...  {% include 'filtros.html' with encabezado="Uso de Filtros" %}  ... |
| --- |

En **filtros.html**

| <br>  <hr>  <br>  <strong> {{encabezado}}</strong>  Filtro<br>  <p> Cadena Original: {{encabezado|default:"Sección sin título"}} </p>  <p> Uso join: {{encabezado|join:'-'}} </p>  <p> Tamaño de cadena: {{encabezado|length}} </p>  <p> Mayúscula: {{encabezado|upper}} </p>  <p> Minúscula: {{encabezado|lower}} </p>  <p> Contador de Palabras: {{encabezado|wordcount}} </p>  <br>  <p>Lista Original: {{estudiantes}}</p>  <p>Uso join: {{estudiantes|join:' - '}}</p>  <p>Primero de la lista: {{estudiantes|first}}</p>  <p>Último de la lista: {{estudiantes|last}}</p>  <p>Random: {{estudiantes|random}}</p>  <p>Tamaño de la Lista: {{estudiantes|length}}</p> |
| --- |

**Observación**: Pueden consultar en <https://docs.djangoproject.com/en/4.0/ref/templates/builtins/>

1. Para crear filtros personalizados:
   1. En la carpeta miapp, crear una carpeta: **templatetags**
   2. Dentro de **templatetags**, crear los archivos:
      1. \_\_init\_\_.py
      2. filters.py



En **filters.py**

| from django import template  register = template.Library()  @register.filter(name='saludo')  def saludo(valor):  return f"<h1 style='background:green; color:white;'> Bienvenido, {valor} </h1>" |
| --- |

En **saludo.html**

| ...  {% block content %}  <h1 class="title">Bienvenidos</h1>  {{autor\_saludo}}  **{% load filters %}**  **{{"Iniciando curso LP3"|saludo|safe}}**  <p>  {{ block.super }}  </p>  {% endblock %} |
| --- |

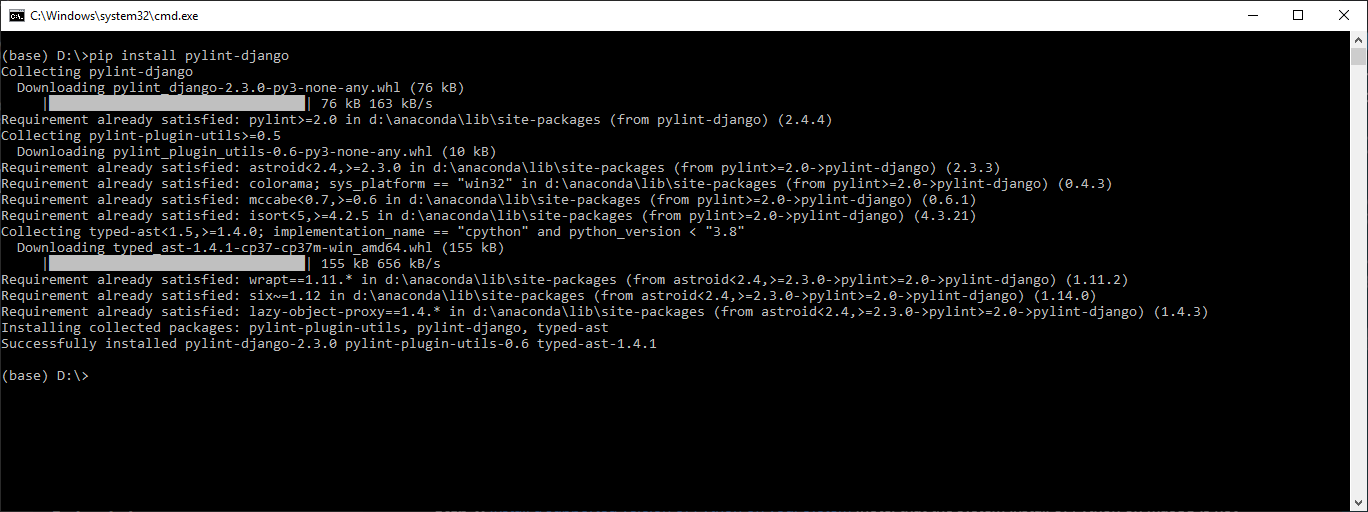
MODELOS Y BASE DE DATOS

Previos

* Instalar Pylint para DJango

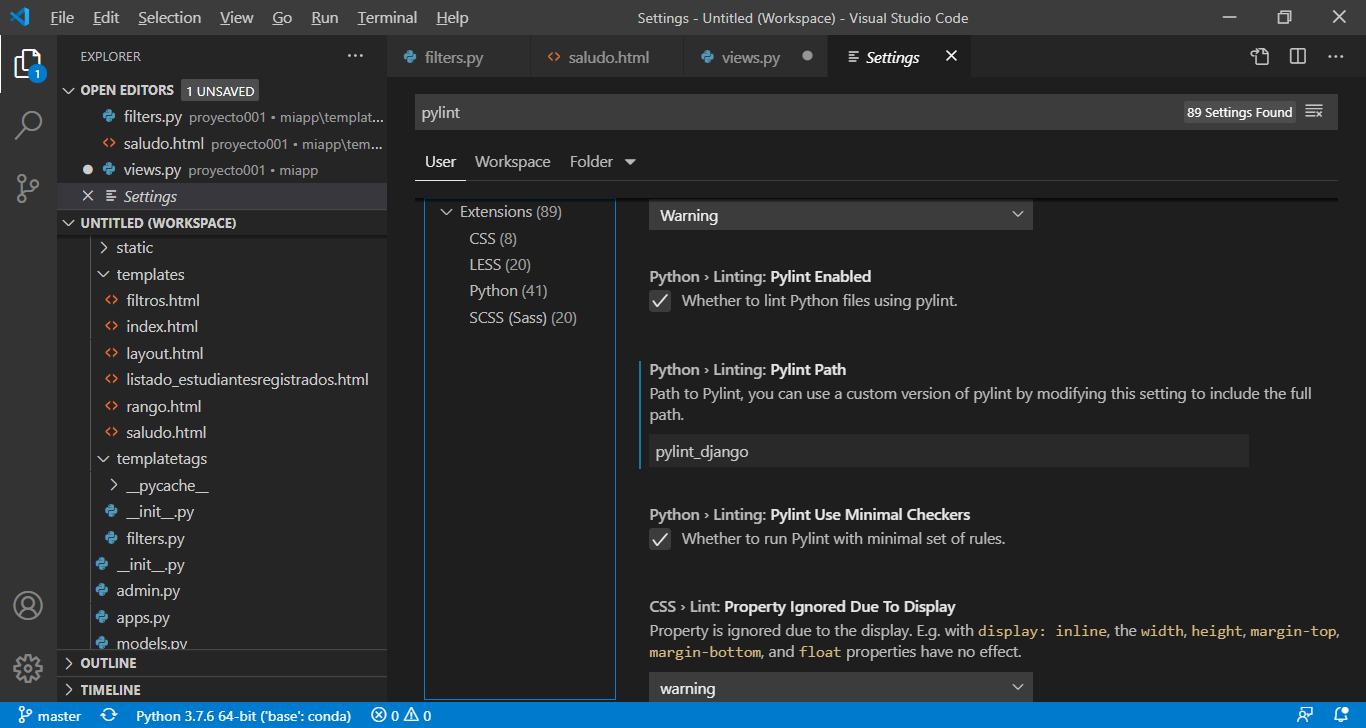
Referencia: <https://pypi.org/project/pylint-django/>

Para instalar: **pip install pylint-django**



* Reiniciamos el VSCode
* Clic File→ Preferences → Setting → Extension → pylint

En Pylint path, reemplazamos, pylint por pylint\_django



**Django**

**Capa de Modelo y Base de Datos en DJango**

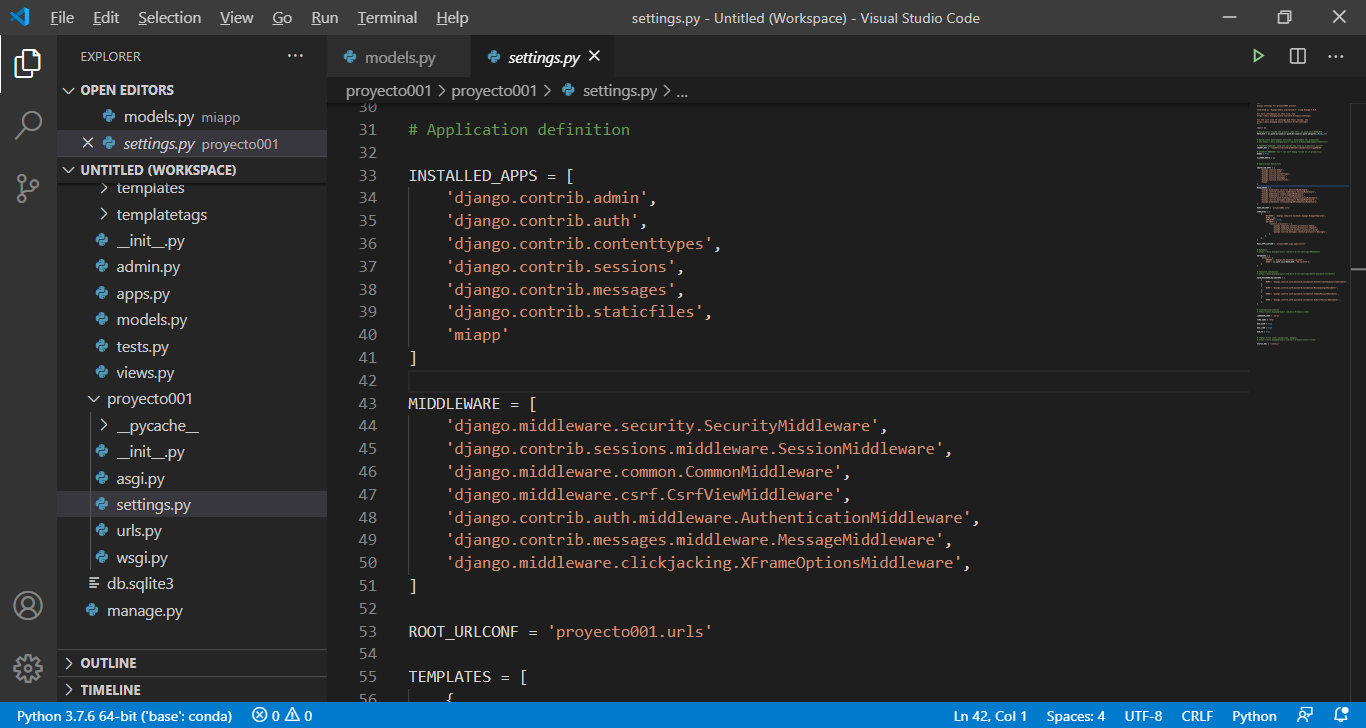
1. Creamos una clase en models.py

| from django.db import models  # Create your models here.  class Articulo(models.Model):  titulo = models.CharField(max\_length=100)  contenido = models.TextField()  publicado = models.BooleanField()  # auto\_now\_add me permitirá registrar  # la fecha cuando cree el registro  creado = models.DateTimeField(auto\_now\_add=True)  # auto\_now me permitirá registrar  # la fecha cuando se modifique el registro  actualizado = models.DateTimeField(auto\_now=True)  class Categoria(models.Model):  nombre = models.CharField(max\_length=100)  descripcion = models.CharField(max\_length=250)  # DateField() para guardar la fecha manualmente  creado = models.DateField() |
| --- |

Importante: Para ver todos los tipos de fields:

<https://docs.djangoproject.com/en/4.0/ref/models/fields/>

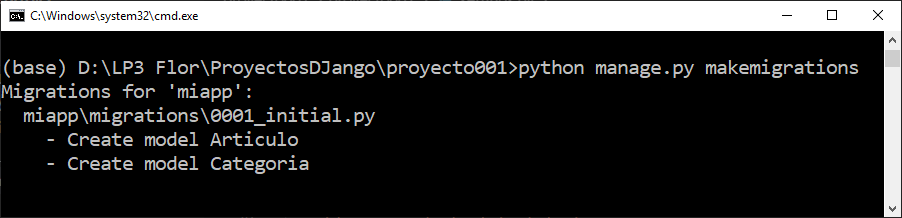
1. ¿Cómo convertir mis clases a tablas?
   1. Considerar que ‘miapp’ esté en **proyecto001/settings.py**



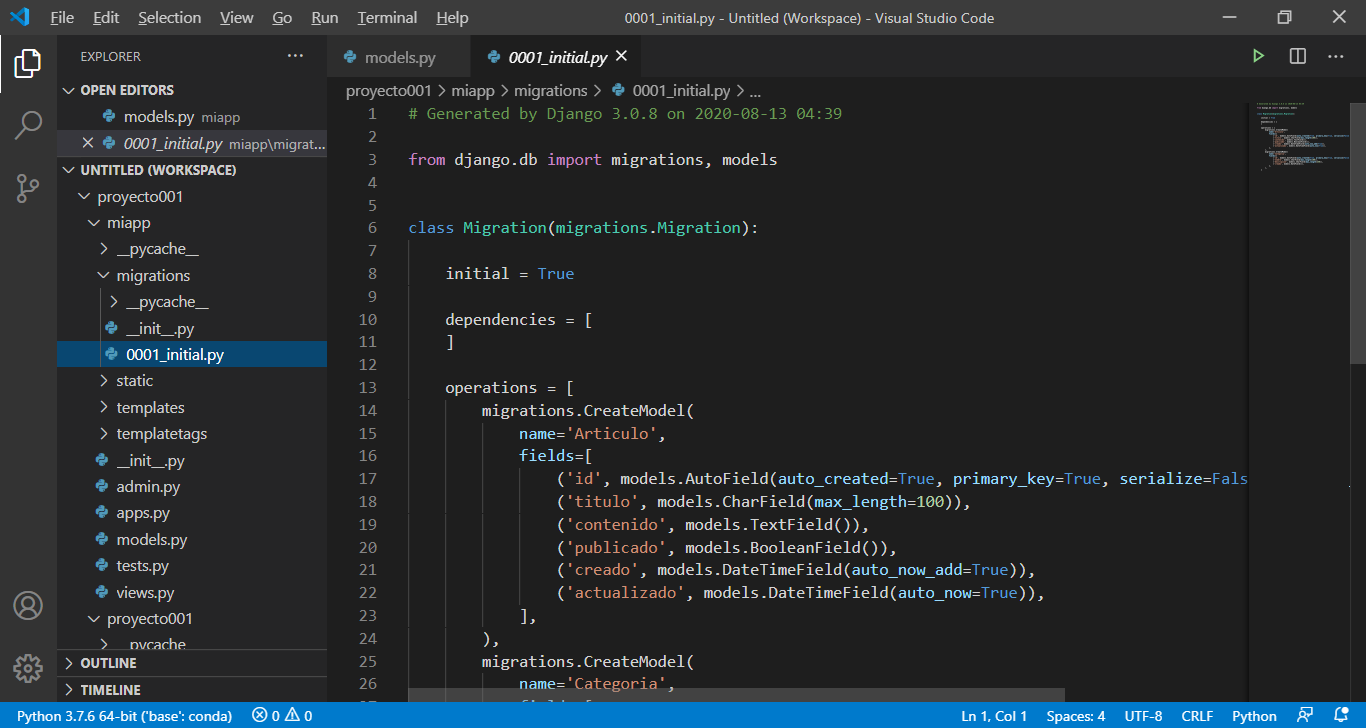
* 1. Tenemos que crear una migración en **migrations**

Migración se ejecuta cuando se hace un cambio en la base de datos, es una forma de versionar la base de datos.

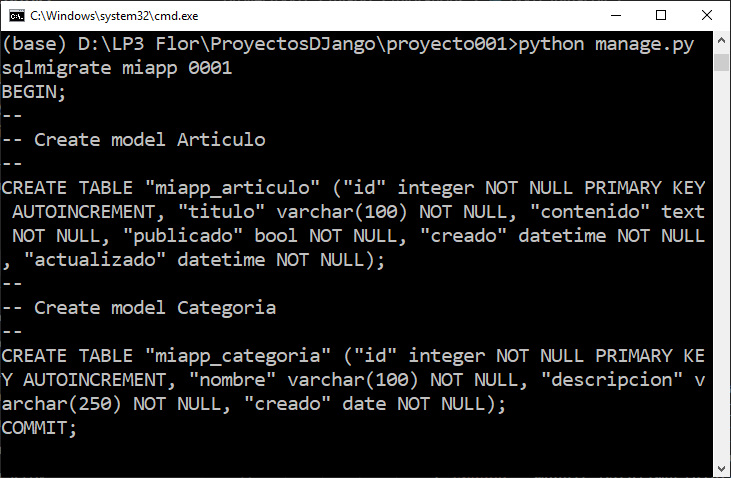
1. Para hacer una migración
   1. En la consola hacemos: **“python manage.py makemigrations”**



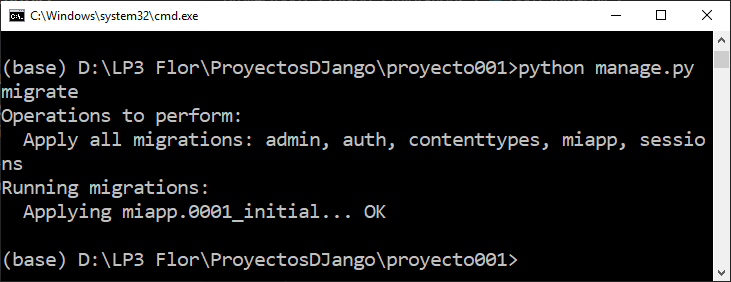
* 1. Se crearon un archivo: 0001\_initial.py



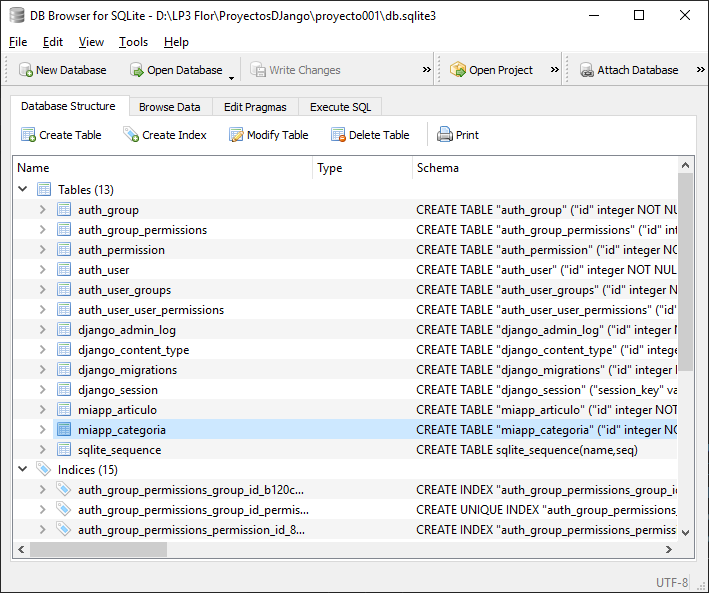
* 1. Para generar la consulta de creación de mis tablas SQL: **python manage.py sqlmigrate miapp 0001**



* 1. Para generar mis tablas en SQLite, hacemos: **python manage.py migrate**



* 1. Ahora podemos abrir el archivo: db.sqlite3



**Recuerde**: DJango te crea un id autoincremental.

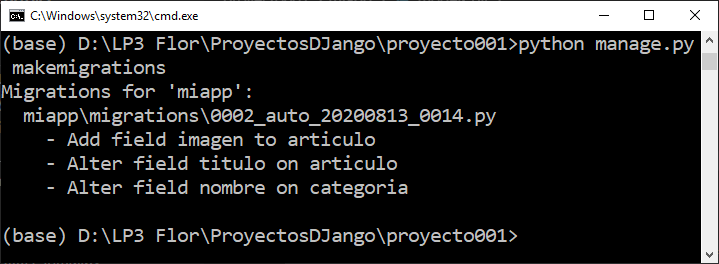
1. Para hacer cambios en los campos de mis tablas, también lo podemos hacer desde DJango,

En este caso:

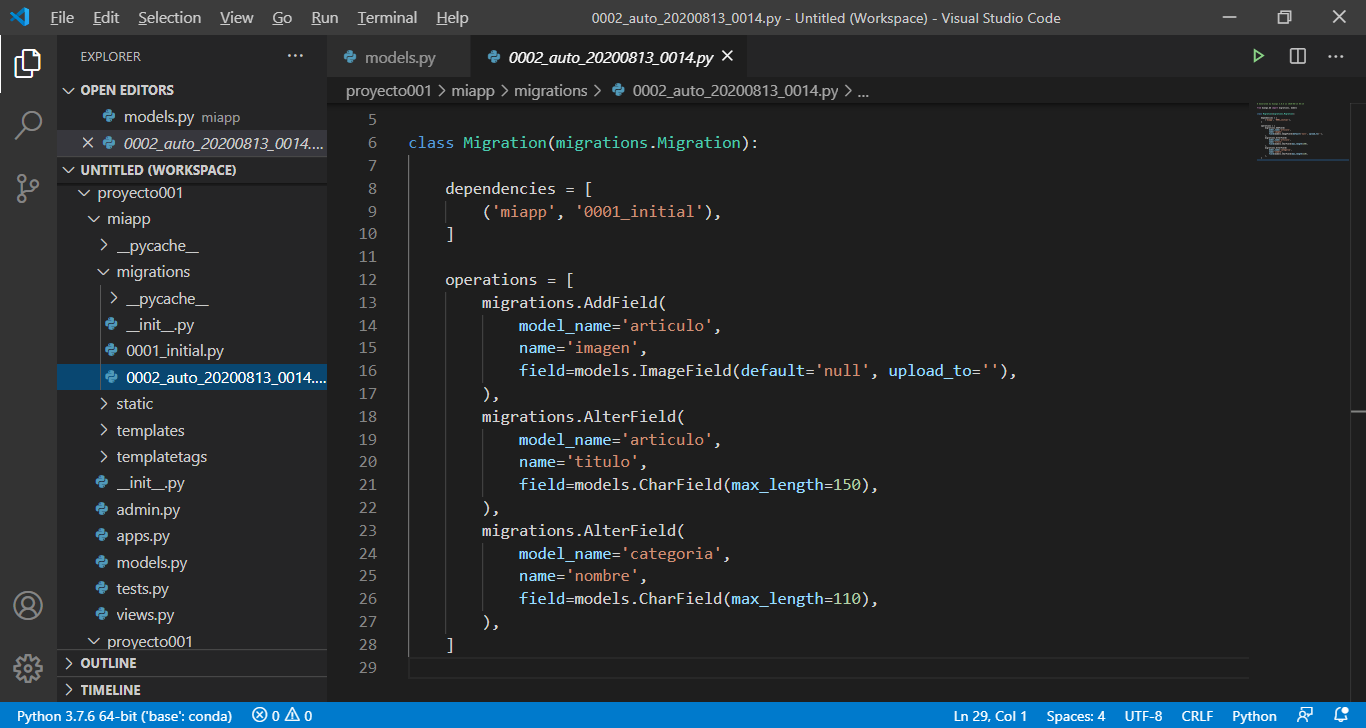
* Agregaremos el campo image
* Modificaremos el tamaño de title en la clase Articulo a 150
* Modificaremos el tamaños de name en la clase Categoria a 110
  1. En models.py

| from django.db import models  # Create your models here.  class Articulo(models.Model):  titulo = models.CharField(max\_length=150)  contenido = models.TextField()  imagen = models.ImageField(default='null')  publicado = models.BooleanField()  # auto\_now\_add me permitirá registrar  # la fecha cuando cree el registro  creado = models.DateTimeField(auto\_now\_add=True)  # auto\_now me permitirá registrar la fecha  # cuando se modifique el registro  actualizado = models.DateTimeField(auto\_now=True)  class Categoria(models.Model):  nombre = models.CharField(max\_length=110)  descripcion = models.CharField(max\_length=250)  # DateField() para guardar la fecha manualmente  creado = models.DateField() |
| --- |

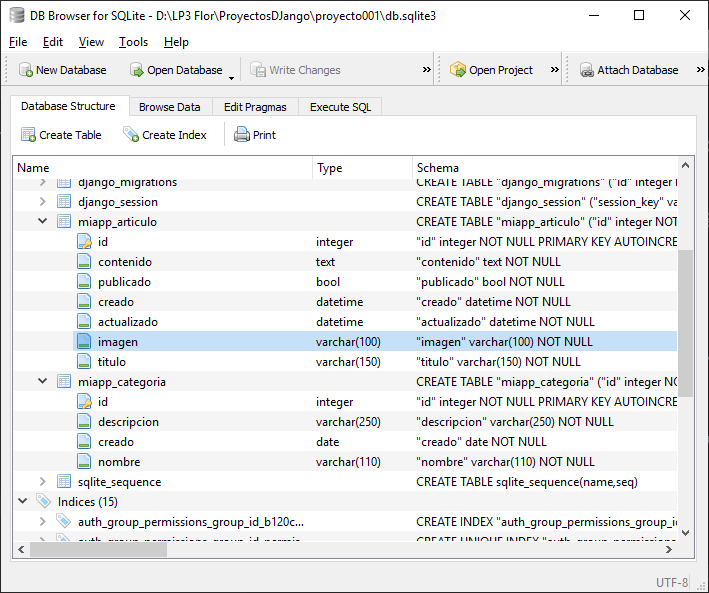
* 1. En la consola: python manage.py makemigrations



* 1. Si observamos en 0002\_auto\_



* 1. Luego para generar la consulta sqlite: **python manage.py sqlmigrate miapp 0002**
  2. Finalmente, para ejecutar los cambios en la base de datos: **python manage.py migrate**



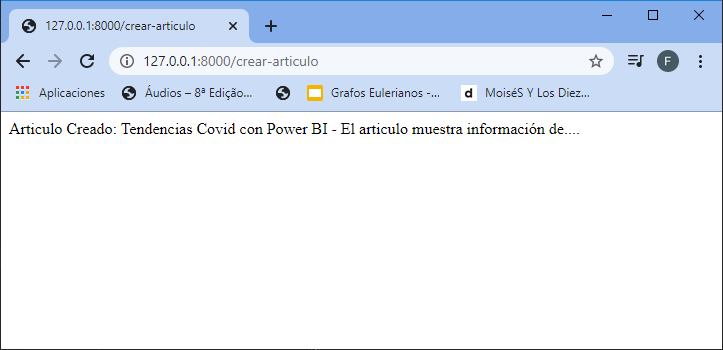
1. Para guardar datos en la base de datos
   1. En views.py

| from django.shortcuts import render, HttpResponse, redirect  **from miapp.models import Articulo**  ...  ...  def crear\_articulo(request):  articulo = Articulo(  titulo = "Tendencias Covid con Power BI",  contenido = "El articulo muestra información de....",  publicado = True  )  articulo.save()  return HttpResponse(f"Articulo Creado: {articulo.titulo} - {articulo.contenido}") |
| --- |

* 1. En urls, creamos una ruta:

| ...  ...  path('crear-articulo',views.crear\_articulo,name="crear\_articulo"), |
| --- |

* 1. Al cargar la página mostrará





1. Guardar registros utilizando parámetros

En urls.py

| ...  ...  path('rango2/<int:a>',views.rango2,name="rango2"),  path('rango2/<int:a>/<int:b>',views.rango2,name="rango2"),  **path('crear-articulo/<str:titulo>/<str:contenido>/<str:publicado>',views.crear\_articulo,name="crear\_articulo"),**  **]** |
| --- |

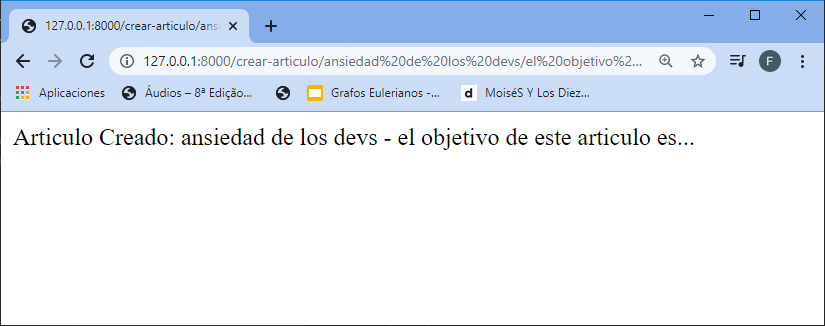
En views.py

| def crear\_articulo(request,titulo, contenido, publicado):  articulo = Articulo(  titulo = titulo,  contenido = contenido,  publicado = publicado  )  articulo.save()  return HttpResponse(f"Articulo Creado: {articulo.titulo} - {articulo.contenido}") |
| --- |

Al cargar: <http://127.0.0.1:8000/crear-articulo>/

Añadimos: **Ansiedad de los devs/- el objetivo de este articulo es /True.**

**Nos quedaría:** <http://127.0.0.1:8000/crear-articulo>/Ansiedad de los devs/el objetivo de este articulo es /True



1. Para buscar un registro de la base de datos

En urls.py

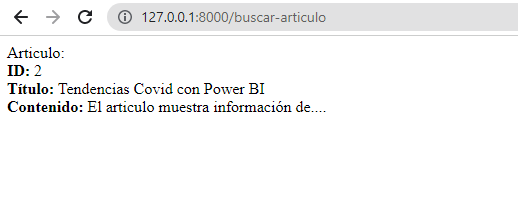
| ...  ...  **path('crear-articulo/<str:titulo>/<str:contenido>/<str:publicado>',views.crear\_articulo,name="crear\_articulo"),**    **path('buscar-articulo', views.buscar\_articulo, name="buscar\_articulo"),** |
| --- |

En views.py

| def buscar\_articulo(request):  try:  articulo = Articulo.objects.get(id=1000)  resultado = f"""Articulo:  <br> <strong>ID:</strong> {articulo.id}  <br> <strong>Título:</strong> {articulo.titulo}  <br> <strong>Contenido:</strong> {articulo.contenido}  """  except:  resultado = "<h1> Artículo No Encontrado </h1>"  return HttpResponse(resultado) |
| --- |

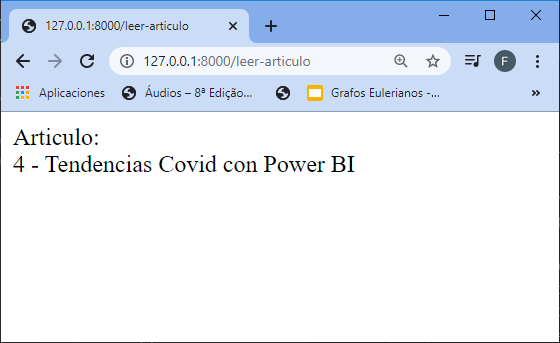
Al cargar: Para esto cambiamos en el código anterior el id= 11, también podría usar (pk=11)

| def buscar\_articulo(request):  try:  articulo = Articulo.objects.get(id=11)  resultado = f"""Articulo:  <br> <strong>ID:</strong> {articulo.id}  <br> <strong>Título:</strong> {articulo.titulo}  <br> <strong>Contenido:</strong> {articulo.contenido}  """  except:  resultado = "<h1> Artículo No Encontrado </h1>"  return HttpResponse(resultado) |
| --- |



Al Cargar con id=1000





**Ejercicio:**

Permitir que se haga la búsqueda enviando parámetros por la URL

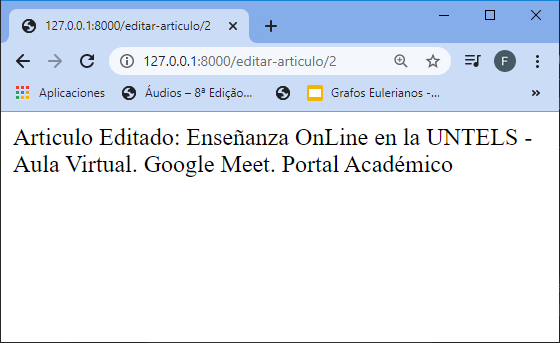
1. Para editar un registro de la tabla articulo

En urls.py

| ...  ...  **path('editar-articulo/<int:id>',views.editar\_articulo,name="editar\_articulo"),** |
| --- |

En views.py

| ...  ...  def editar\_articulo(request, id):  articulo = Articulo.objects.get(pk=id)  articulo.titulo = "Enseñanza onLine en la UNTELS"  articulo.contenido = "Aula Virtual, Google Meet, Portal Académico, Google Classroom..."  articulo.publicado = False  articulo.save()  return HttpResponse(f"Articulo Editado: {articulo.titulo} - {articulo.contenido}") |
| --- |



1. Para listar todos los registros de la tabla articulo

En views.py

| ...  ...  def listar\_articulos(request):  articulos = Articulo.objects.all()  return render(request, 'listar\_articulos.html',{  'articulos':articulos,  'titulo': 'Listado de Artículos'  }) |
| --- |

Creamos el template: templates/listar\_articulos.html

| {% extends 'layout.html' %}  {% block title %}  {{titulo}}  {% endblock %}  {% block content %}  <h1 class="title">{{titulo}} </h1>  <ul>  {% for articulo in articulos %}  <li>  <h4>{{articulo.id}} - {{articulo.titulo}}</h4>  <span>Creado: {{articulo.creado}}</span>  {% if articulo.publicado %}  <strong> Publicado </strong>  {% else %}  <strong> No Publicado/En Revisión </strong>  {% endif %}  <p>{{articulo.contenido}}</p>  </li>  {% endfor %}  </ul>  {% endblock %} |
| --- |

En urls.py

| ...  ...  path('listar-articulos/',views.listar\_articulos,name="listar\_articulos"), |
| --- |



En el Wiew.py podemos ver el ordenamiento

| ef listar\_articulos(request):  articulos = Articulo.objects.all()  return render(request, 'listar\_articulos.html',{  'articulos':articulos,  'titulo': 'Listado de Artículos'  }) |
| --- |

**01 Importante** (limit y order by):

| articulos = Articulo.objects.order\_by('titulo') | Ordena por título |
| --- | --- |
| articulos = Articulo.objects.order\_by('-titulo') | Ordena por título en orden descendente |
| articulos = Articulo.objects.order\_by('-titulo')[:3] | Muestra 3 artículos ordenado por título en forma descendente |
| articulos = Articulo.objects.order\_by('-titulo')[3:6] | Muestra los artículos de la posición 3 al 6 ordenado por título en forma descendente |

**02 Importante**: Para configurar el idioma de la fecha, en **proyecto001/settings.py**, asignamos:

| LANGUAGE\_CODE = 'es-es' |
| --- |

1. Para eliminar registros de la tabla articulo

En views.py

| ...  ...  def eliminar\_articulo(request, id):  articulo = Articulo.objects.get(pk=id)  articulo.delete()  return redirect('listar\_articulos') |
| --- |

En urls.py

| ...  ...  path('eliminar-articulo/<int:id>',views.eliminar\_articulo, name='eliminar\_articulo') |
| --- |

En templates/listar\_articulos.html

| ...  {% for articulo in articulos %}  <li>  <h4>{{articulo.id}} - {{articulo.titulo}}</h4>  <span>Creado: {{articulo.creado}}</span>  {% if articulo.publicado %}  <strong> Publicado </strong>  {% else %}  <strong> No Publicado/En Revisión </strong>  {% endif %}  <p>  {{articulo.contenido}}<br>  **<a href="{% url 'eliminar\_articulo' id=articulo.id %}">Eliminar</a>**  </p>  </li>  {% endfor %}  ... |
| --- |

1. Finalmente en layout, agregamos la opción **Artículo**

| <li>  <a href="{% url 'listar\_articulos' %}"> Artículos</a>  </li> |
| --- |



1. Uso de filter y lookups

En **views.py**, dentro del **def listar\_articulos()**

| articulos = Articulo.objects.filter(titulo="ansiedad de los devs") | Devuelve todos los articulos cuyo titulo sea ... |
| --- | --- |
| articulos = Articulo.objects.filter(titulo="ansiedad de los devs",id=6) | Devuelve todos los articulos cuyo titulo sea … y cuyo id = 6 |
| articulos = Articulo.objects.filter(titulo\_\_contains="BI") | Devuelve todos los articulos cuyo titulo contenga la palabra: **BI** |
| articulos = Articulo.objects.filter(titulo\_\_exact="BI") | Devuelve todos los articulos cuyo titulo contenga sea exactamente: **BI** |
| articulos = Articulo.objects.filter(titulo\_\_iexact="BI") | Devuelve todos los articulos cuyo titulo contenga sea exactamente (Sin considerar mayúsculas o minúsculas): **BI** |
| articulos = Articulo.objects.filter(id\_\_gt=6) | Mayor que |
| articulos = Articulo.objects.filter(id\_\_gte=6) | Mayor o igual que |
| articulos = Articulo.objects.filter(id\_\_lt=6) | Menor que |
| articulos = Articulo.objects.filter(id\_\_gt=6, titulo\_\_contains="BI") | Menor o igual que 6 **AND** el título contiene la palabra BI |

| articulos = Articulo.objects.filter(  titulo\_\_contains="Python"  ).exclude(  publicado = False  ) | **exclude**  Los artículos que contengan el título “**Java**” y excluya los artículos cuyo estado de publicación sea **False**. |
| --- | --- |
| articulos = Articulo.objects.raw(""" select \* from miapp\_articulo  where  id >= 6 """) | **Consultas SQL**  También podemos ejecutar consultas SQL |
| ...  from django.db.models import Q  ...  ...  articulos = Articulo.objects.filter(  Q(titulo\_\_contains="BI") | Q(titulo\_\_contains="Python")  ) | **Para utilizar el OR (Lógico)**  Primero importamos la librería Q  Luego utilizamos Q en la consulta |

**FORMULARIOS EN DJANGO**

1. Primeramente,

En views.py

| ...  ...  def save\_articulo(request):  articulo = Articulo(  titulo = titulo,  contenido = contenido,  publicado = publicado  )  articulo.save()  return HttpResponse(f"Articulo Creado: {articulo.titulo} - {articulo.contenido}")  def create\_articulo(request):  return render(request, 'create\_articulo.html')  ...  ... |
| --- |

En template, Creamos **create\_articulo.hml**

| {% extends 'layout.html' %}  {% load filters %}  {% block title %}  Formulario en DJango  {% endblock %}  {% block content %}  <h1 class="title">Formularios en DJango</h1>    {% endblock %} |
| --- |

En **urls.py**

| path('save-articulo/',views.save\_articulo, name='save\_articulo'),  path('create-articulo/',views.create\_articulo, name='create\_articulo'), |
| --- |

En **layout.html**

| ...  <li>  <a href="{% url 'listar\_articulos' %}"> Artículos</a>  </li>  **<li>**  **<a href="{% url 'create\_articulo' %}"> Crear Artículo</a>**  **</li>**  <li>  <a href="{% url 'saludo' %}"> Mensaje de Saludo</a>  </li>  ... |
| --- |



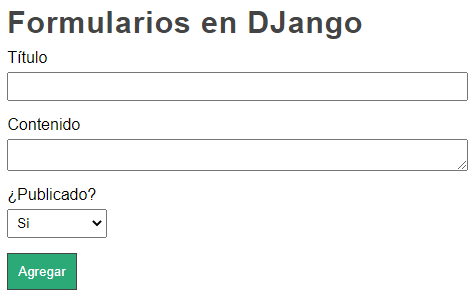
1. Diseño de un formulario de registro de artículos

En **create\_articulo.html**

| {% block content %}  <h1 class="title">Formularios en DJango</h1>  **<form action="{% url 'save\_articulo' %}" method="GET">**  **<label for="titulo">Título</label>**  **<input type="text" name="titulo" />**  **<label for="contenido">Contenido </label>**  **<textarea name="contenido" ></textarea>**  **<label for="publicado">¿Publicado?</label>**  **<select name="publicado">**  **<option value="1"> Si </option>**  **<option value="0"> No </option>**  **</select>**  **<input type="submit" value="Agregar" />**  **</form>**    {% endblock %} |
| --- |

1. Para agregar estilos al formulario, editamos **styles.css**, para ello nos vamos a la zona de los estilos para **.box**

| ...  ...  .box{  background: white;  width: 95%;  min-height: 930px;  padding: 20px;  border: 1px solid #ddd;  border-radius: 2px;  margin: 0 auto;  }  **.box form{**  **width: 40%;**  **}**  **.box form input, .box form label{**  **display: block;**  **padding: 5px;**  **padding-left: 0px;**  **}**  **.box form input[type="text"],**  **.box form textarea,**  **.box form select{**  **width: 100%;**  **margin-bottom: 10px;**  **}**  **.box form select{**  **width: 100px;**  **padding: 5px;**  **}**  **.box form input[type="submit"],**  **.box form input[type="button"],**  **.box form button{**  **padding: 10px;**  **margin-top: 5px;**  **background: #2ba977;**  **border: 1px solid #444;**  **color: white;**  **transition: 300ms all;**  **}**  **.box form input[type="submit"]:hover,**  **.box form input[type="button"]:hover,**  **.box form button:hover{**  **cursor: pointer;**  **background: #1f7e58;**  **}**  ... |
| --- |



1. Al ingresar valores y al hacer clic en el botón agregar, mostrará el siguiente error. Porque el métodos save\_articulo, no está recibiendo los valores.



1. En views.py, dentro de save\_articulo

| …  def save\_articulo(request):  if request.method == 'GET':  titulo = request.GET['titulo']  contenido = request.GET['contenido']  publicado = request.GET['publicado']  articulo = Articulo(  titulo = titulo,  contenido = contenido,  publicado = publicado  )  articulo.save()  return HttpResponse(f"Articulo Creado: {articulo.titulo} - {articulo.contenido}")  else:  return HttpResponse("<h2>No se ha podido registrar el artículo</h2>")    … |
| --- |

y para que muestre todo el listado en Views.py → listar\_articulos(), deshabilitamos los filtros que vimos la sesión anterior y listamos todo

| def listar\_articulos(request):  articulos = Articulo.objects.all();  """articulos = Articulo.objects.filter(  Q(titulo\_\_contains="Py") |  Q(titulo\_\_contains="Hab")  )"""  return render(request, 'listar\_articulos.html',{  'articulos': articulos,  'titulo': 'Listado de Artículos'  }) |
| --- |

1. ¿Qué pasaría si existe una restricción, que indica que los títulos deben ser mayores a 5 caracteres.

| ...  def save\_articulo(request):  if request.method == 'GET':  titulo = request.GET['titulo']  if len(titulo)<=5:  return HttpResponse("<h2>El tamaño del título es pequeño, intente nuevamente</h2>")  contenido = request.GET['contenido']  publicado = request.GET['publicado']  ... |
| --- |

1. Pero, utilizar el método GET no es muy seguro, en ese caso es mejor utilizar el método POST, entonces

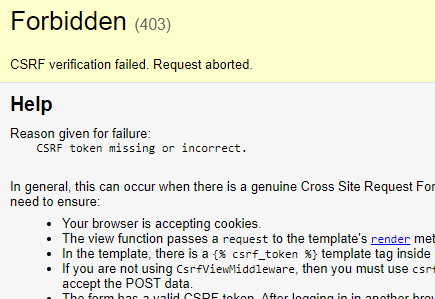
En create\_articulo.html, cambiamos GET por POST

| <form action="{% url 'save\_articulo' %}" name="" method="POST">  <label for="titulo">Título</label>  <input type="text" name="titulo" /> |
| --- |

En views.py, tambien cambiamos GET por POST

| ...  def save\_articulo(request):  if request.method == 'POST':  titulo = request.POST['titulo']  if len(titulo)<=5:  return HttpResponse("<h2>El tamaño del título es pequeño, intente nuevamente</h2>")  contenido = request.POST['contenido']  publicado = request.POST['publicado']  …  … |
| --- |

Entonces, me va a generar un nuevo error, indicando que la verificación CSRF ha fallado:



Si observamos la 3ra observación, nos recomienda agregar el csrf\_token, para ellos en create\_articulo:

| ...  <form action="{% url 'save\_articulo' %}" name="" method="POST">  {% csrf\_token %}  <label for="titulo">Título</label>  <input type="text" name="titulo" />  ... |
| --- |

De esta manera, ya tenemos el envío de datos de nuestro formulario utilizando el método POST.

Ejercicio:

Utilizando método POST

* Crear la tabla
  + Autor
    - nombre
    - apellido
    - sexo
    - Fecha Nacimiento
    - Pais
  + Crear los accesos
    - Autor: Que liste los autores y tenga la opción de eliminar
    - Crear Autor: Considere un formulario para registrar a un autor, considere Sexo y Pais con un <select>

**Formulario Basado en Clases**

Django trae objetos en la capa de abstracción para generar formularios de una manera más ágil. Esto se logrará creando formularios en base a clases de python.

1. En **miapp**, creamos el archivo: **forms.py**
2. En **forms.py**, registramos

Para los fields se puede consultar: <https://docs.djangoproject.com/en/4.0/ref/models/fields>

| from django import forms  class FormArticulo(forms.Form):  titulo = forms.CharField(  label="Titulo"  )  # Usar CharField para generar un campo de texto normal  contenido = forms.CharField(  label = "Contenido",  widget = forms.Textarea  ) |
| --- |

1. En **views.py**, agregamos lo siguiente:

| …  from miapp.models import Articulo  from django.db.models import Q  from miapp.forms import FormArticulo  …  …  def create\_full\_articulo(request):  formulario = FormArticulo()  return render(request, 'create\_full\_articulo.html',{  'form': formulario  }) |
| --- |

1. Ahora, en **templates**, creamos el archivo: **create\_full\_articulo.html**, luego copiamos todo el contenido de **create\_articulo.html** y lo pegamos en **create\_full\_articulo.html**, luego eliminamos el contenido de **<form>** y eliminamos el contenido del **action**.

| {% extends 'layout.html' %}  {% load filters %}  {% block title %}  Formulario en DJango  {% endblock %}  {% block content %}  <h1 class="title">  Formularios en DJango  </h1>  <form action="" method="POST">  {% csrf\_token %}  {{form}}    <input type="submit" value="Agregar" />  </form>  {% endblock %} |
| --- |

1. En URL, creamos el path:

| path('create-full-articulo/',views.create\_full\_articulo, name='create\_full\_articulo'), |
| --- |

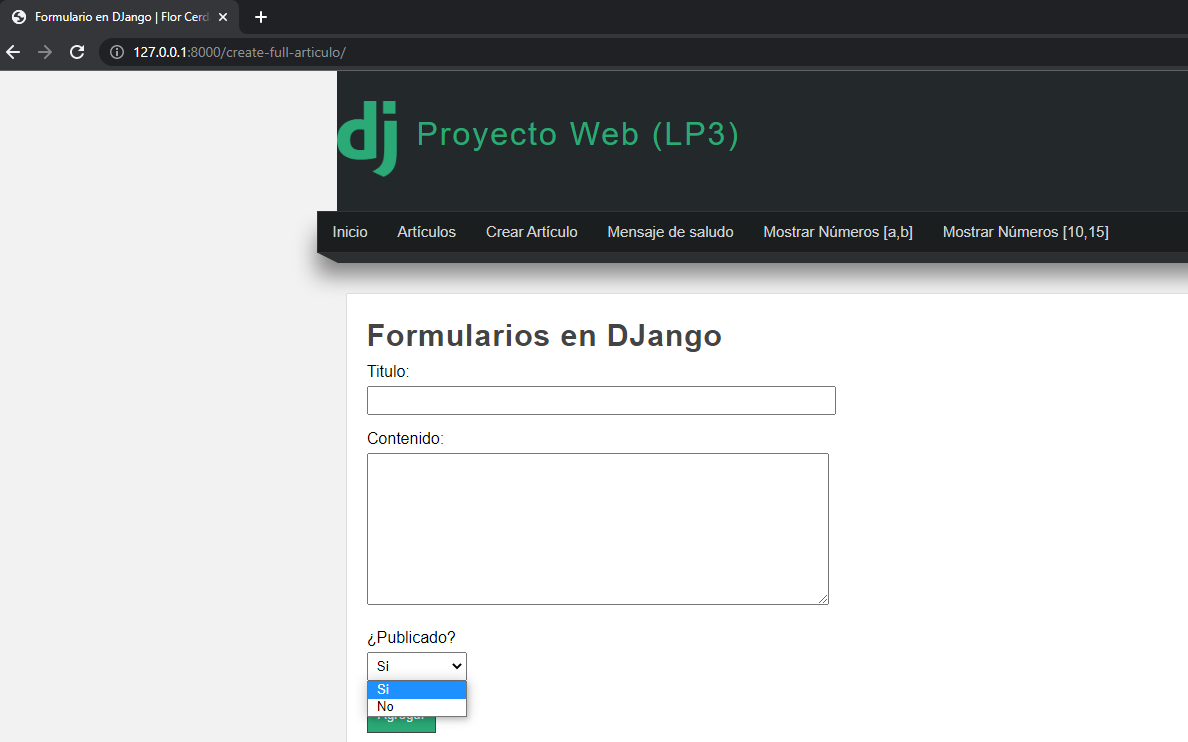
1. En esta modalidad, también puedo incluir cada uno de los elementos dentro de una etiqueta. Entonces en el **form** del **create\_full\_articulo.html**:

| {{form.as\_p}} | Me ubica cada uno de los elementos del formulario en un párrafo |
| --- | --- |
| {{ form.as\_ul }} | Me ubica cada uno de los elementos del formulario en un viñetas |

Lo dejamos con **{{form.as\_p}}**

1. Ahora, para hacer el <select> en forms.py

| from django import forms  class FormArticulo(forms.Form):  titulo = forms.CharField(  label="Titulo"  )  # Usar CharField para generar un campo de texto normal  contenido = forms.CharField(  label = "Contenido",  widget = forms.Textarea  )    opciones\_publicado = [  (1, 'Si'),  (0, 'No'),  ]  publicado = forms.TypedChoiceField(  label = "¿Publicado?",  choices = opciones\_publicado  ) |
| --- |



1. Ahora, para recibir la información de este formulario, en **views.py**,

| def create\_full\_articulo(request):  if request.method == 'POST':  formulario = FormArticulo(request.POST)  if formulario.is\_valid():  data\_form = formulario.cleaned\_data  # Hay 2 formas de recuperar la información  titulo = data\_form.get('titulo')  contenido = data\_form['contenido']  publicado = data\_form['publicado']  return HttpResponse(titulo + ' - ' + contenido + ' - ' + str(publicado))  else:  formulario = FormArticulo()  # Generamos un formulario vacío  return render(request, 'create\_full\_articulo.html',{  'form': formulario  }) |
| --- |

1. ¿Cómo puedo hacer para guardar la información?, copiamos lo que hicimos en **def save\_articulo()**

| if formulario.is\_valid():  data\_form = formulario.cleaned\_data  # Hay 2 formar de recuperar la información  titulo = data\_form.get('titulo')  contenido = data\_form['contenido']  publicado = data\_form['publicado']  articulo = Articulo(  titulo = titulo,  contenido = contenido,  publicado = publicado  )  articulo.save()  return HttpResponse(articulo.titulo + ' - ' + articulo.contenido + ' - ' + str(articulo.publicado)) |
| --- |

1. Ahora, si queremos que al crear el artículo, te redirija a la página de artículos, comentamos nuestro **HttpResponse** y creamos un **redirect**

| …  …  articulo = Articulo(  titulo = titulo,  contenido = contenido,  publicado = publicado  )  articulo.save()  return redirect('listar\_articulos')  #return HttpResponse(articulo.titulo + ' - ' + articulo.contenido + ' - ' + str(articulo.publicado))  …  … |
| --- |

1. Finalmente, en el layout, agregamos el acceso en el menú

| …  <li>  <a href="{% url 'create\_full\_articulo' %}"> Crear Artículo2</a>  </li>  … |
| --- |

PERSONALIZAR NUESTRO FORMULARIO

1. Para agregar algunas validaciones en **forms.py**,

| class FormArticulo(forms.Form):  titulo = forms.CharField(  label="Titulo",  max\_length=40,  required=False,  widget=forms.TextInput(  attrs={  'placeholder': 'Ingrese el título',  'class': 'titulo\_form\_articulo'  }  )    )  # Usar CharField para generar un campo de texto normal  contenido = forms.CharField(  label = "Contenido",  widget = forms.Textarea(  attrs={  'placeholder': 'Ingrese el contenido',  'class': 'contenido\_form\_articulo'  }  )  ) |
| --- |



1. Otra forma de hacer las validaciones son:

| …  contenido = forms.CharField(  label = "Contenido",  widget = forms.Textarea  )  contenido.widget.attrs.update({  'placeholder': 'Ingrese el contenido del artículo',  'class': 'contenido\_form\_articulo',  'id': 'contenido\_form'  })  … |
| --- |

Al Cargar



VALIDACIÓN DE FORMULARIOS EN DJANGO

1. Si tuviera errores en el formulario.

En **create\_full\_articulo.py**

| …  {% block content %}  <h1 class="title">  Formularios en DJango  </h1>    {% if form.errors %}  <strong class="rojo">  Hay errores en el Formulario  </strong>  {% endif %}  <form action="" method="POST">  … |
| --- |

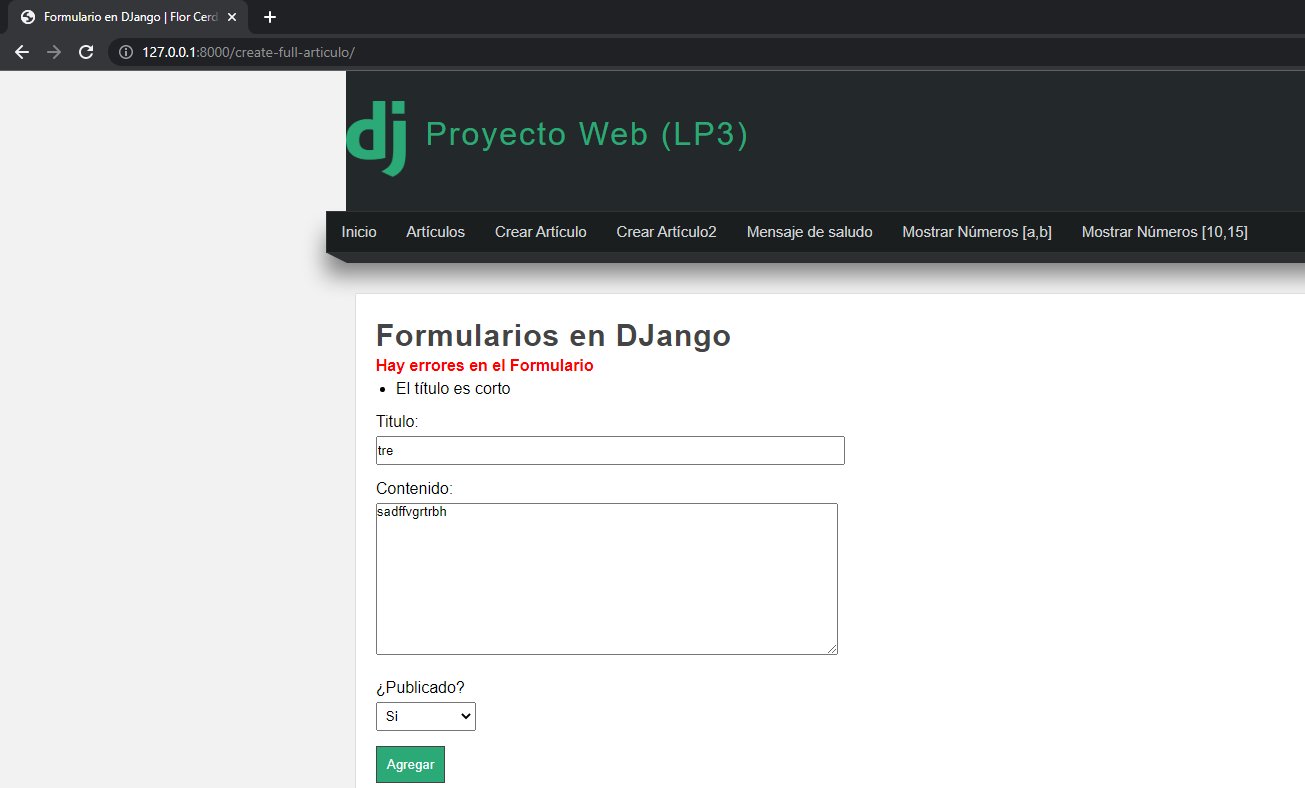
En **style.css**

| …  …  .rojo{  color: red;  }    .box form input[type="submit"]:hover,  .box form input[type="button"]:hover,  … |
| --- |

En forms.py

| from django import forms  from django.core import validators  class FormArticulo(forms.Form):  titulo = forms.CharField(  label="Titulo",  max\_length=40,  required=True,  widget=forms.TextInput(  attrs={  'placeholder': 'Ingrese el título',  'class': 'titulo\_form\_articulo'  }  ),  validators=[  validators.MinLengthValidator(4, 'El título es corto'),  validators.RegexValidator('^[A-Za-z0-9]\*$','El título tiene caracteres inválidos','titulo\_invalido')  ]  ) |
| --- |

Al Cargar



Tambien se puede consultar en: <https://docs.djangoproject.com/en/4.0/ref/validators/>

Ahora, podemos probar que funciona:

* El título es obligatorio
* El título no puede ser menor a 4 caracteres
* El título no permite caracteres especiales
* Sin embargo, tampoco permite espacios entre las palabras del título, para ello, agregamos un espacio en blanco después del 0-9:

| validators.RegexValidator('^[A-Za-z0-9 ]\*$','El título tiene caracteres inválidos','titulo\_invalido') |
| --- |

* Para que permita la ñ y la Ñ, tambien le agregamos

| '^[A-Za-z0-9ñÑ ]\*$' |
| --- |

Ahora, agregamos validaciones para el contenido:

| contenido = forms.CharField(  label = "Contenido",  widget = forms.Textarea,  validators=[  validators.MaxLengthValidator(20,'Superaste el límite de caracteres')  ]  ) |
| --- |

Al Cargar



También, es posible mostrar los errores, desde **create\_full\_articulo.html**

| {% if form.errors %}  <strong class="rojo">  Hay errores en el Formulario  {{form.errors}}  </strong>  {% endif %} |
| --- |

Al cargar



**MENSAJE FLASH**

¿Qué pasaría si deseo mostrar un mensaje que solo dure una sesión de pantalla?

1. En **views.py**, dentro del método: **create\_full\_articulo()**

| from django.contrib import messages  …  articulo.save()  # Crear un mensaje flash (Sesión que solo se muestra 1 vez)  messages.success(request, f'Se agregó correctamente el artículo {articulo.id}')  return redirect('listar\_articulos')  #return HttpResponse(articulo.titulo + ' - ' + articulo.contenido + ' - ' + str(articulo.publicado))  … |
| --- |

En listar\_articulos.html

| …  {% block content %}  <h1 class="title"> {{titulo}} </h1>  {% if messages %}  {% for message in messages %}  <div class="message">  {{message}}  </div>  {% endfor %}  {% endif %}  … |
| --- |

Finalmente le agregamos el estilo en **styles.css**

| .rojo{  color: red;  box-shadow: 0px 0px 4px black;  }  .message{  padding: 20px;  background: #1f7e58;  color: white;  text-align: center;  margin-bottom: 10px;  } |
| --- |

Al cargar

