

Esta clase va a ser

- grabada
- a

Clase 21. PYTHON

Playground Intermedio Parte III

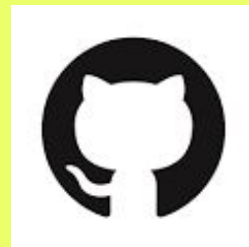
Objetivos de la clase

- **Aprender** a manejar un formulario
- **Crear** formularios para insertar datos
- **Crear** formularios de búsqueda en la BD.

Repositorio Github

Te dejamos el acceso al Repositorio de Github donde encontrarás todo el material complementario y scripts de la clase.

✓ [Repositorio Python](#)



Temario

20

Playground Intermedio Parte II

- ✓ Profundizando en MVT
- ✓ Visitas
- ✓ URLs

21

Playground Intermedio Parte III

- ✓ [Formularios](#)
- ✓ [Creación de formularios](#)
- ✓ [Búsquedas con Form](#)

22

Playground Avanzado Parte I

- ✓ CRUD
- ✓ Clases basadas en vistas

Formularios

Formularios


Hasta el momento sabemos insertar datos por medio de la terminal (**ejemplo: `curso.save()`**) y por medio del panel de administración (**ejemplo: por medio del `add`**).

Básicamente, con esas dos alternativas estamos haciendo lo que se llama un “insert” en nuestra base de datos. Pero ninguna de estas dos opciones suelen ser las más utilizadas.

Formularios

Lo más utilizado suele ser los formularios, que no es otra cosa más que una vista que permite poder agregar datos a nuestro model directamente desde nuestra web.

Estamos seguro de que ya saben que es un formulario:



The image shows a web form titled "Float Labels" set against a purple-to-pink gradient background. The form is a white rounded rectangle containing four input fields with labels that float above them: "Nombre" (with a cursor in the field), "Email", "Edad", and "Mensaje". At the bottom right of the form is a blue button with the text "Suscribete" in white.

Formularios

Los formularios se pueden hacer por medio de HTML con CSS (sabiendo algo de desarrollo web), pero también podemos hacer uso de facilidades que nos da Django para lograrlo.



¿Cómo funcionan los formularios?

¿Cómo funcionan los formularios?

El html recibe nuestra información por medio de la vista y su template asociado. Al apretar un botón esa información viaja por medio de un método **GET** o **POST** y llega al servidor, donde esos datos se manipulan.

Creación de formularios (HTML)



Creación de formularios

1

Vamos a las vistas, views.py y creamos una nueva.

```
def cursoFormulario(request):  
    return render(request,  
        "AppCoder/cursoFormulario.html")
```

2

Luego agregamos la vista a las `urls.py`

```
path('cursoFormulario', views.cursoFormulario,  
    name="CursoFormulario")
```

3

Creamos el `html`. (donde haremos el formulario)

`cursoFormulario.html`



Creación de formularios

4

Vamos a las vistas, `views.py` y creamos una nueva.

```
def cursoFormulario(request):  
    return render(request, "AppCoder/cursoFormulario.html")
```

5

Luego agregamos la vista a las `urls.py`

```
urlpatterns = [  
    path('', views.inicio, name="Inicio"), #esta era nuestra primer view  
    path('cursos', views.cursos, name="Cursos"),  
    path('profesores', views.profesores, name="Profesores"),  
    path('estudiantes', views.estudiantes, name="Estudiantes"),  
    path('entregables', views.entregables, name="Entregables"),  
    path('cursoFormulario', views.cursoFormulario, name="CursoFormulario"),  
]
```

6

Creamos el `html`. (donde haremos el formulario)



Creación de formularios

```
views.py M  cursoFormulario.html U  urls.py ProyectoCoder  admin.py  url
AppCoder > templates > AppCoder > cursoFormulario.html > html > body > form
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Formulario - Agregar Curso</title>
8  </head>
9  <body>
10
11  1) <form action="/cursoFormulario/" method="POST">
12
13
14  2) y 3) <p>Curso: <input type="text" name="curso"></p>
15          <p>Camada: <input type="number" name="camada"></p>
16
17
18  4) <input type="submit" value="Enviar">
19
20
21  </form>
22
23
24
25
26 </body>
27 </html>
```

- 1) Action es el nombre de la url
- 2) Method es la forma en la que se envían los datos
- 3) Input son espacios para escribir información
- 4) Input Submit es el botón que envía la información.

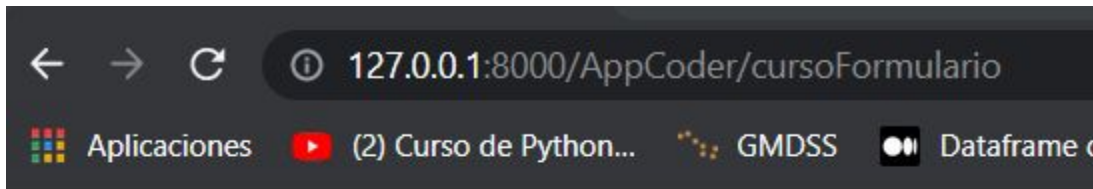


Script

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible"
content="IE=edge">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Formulario - Agregar Curso</title>
</head>
<body>
  <form action="/cursoFormulario/" method="post">
    <p> Curso: <input type = "text" name="curso">
    <p> Camada: <input type = "text" name="camada">
    <input type = "submit" value="Enviar">
  </form>
</body>
</html>
```




Creación de formularios



Curso:

Camada:

Enviar

¡Nuestra vista! 🙌



Creación de formularios

Obviamente, al apretar el botón Enviar, aún no hace nada. Necesitamos que esos datos viajen al servidor y podamos hacer algo con ellos.

Para esto:

1. Debemos agregar el token de validación que nos exige Django
2. En la vista que creamos recibimos los datos.





Creación de formularios

De esta forma podemos guardar en la base de datos los datos recibidos por medio del form.

```
{% csrf_token %}
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Formulario - Agregar Curso</title>
</head>
<body>

  <form action="/cursoFormulario/" method="POST">{% csrf_token %}

    <p>Curso: <input type="text" name="curso"></p>

    <p>Camada: <input type="number" name="camada"></p>

    <input type="submit" value="Enviar">

  </form>

</body>
</html>
```



Creación de formularios

Pero nos podría generar problemas de validación de los datos enviados, así que lo haremos con los formularios de Django.

```
def cursoFormulario(request):  
  
    if request.method == 'POST':  
  
        curso = Curso ( request.POST['curso'], request.POST['camada'])  
  
        curso.save()  
  
        return render(request, "AppCoder/inicio.html")  
  
    return render(request, "AppCoder/cursoFormulario.html")
```



Creación de formularios

Script:

```
def cursoFormulario(request):  
    if request.method == 'POST':  
  
        curso = Curso(request.post['curso'],(request.post['camada']))  
  
        curso.save()  
  
        return render(request, "AppCoder/inicio.html")  
  
        return render(request, "AppCoder/cursoFormulario.html")
```

Creación de formularios (Api from Django)



Creación de formularios (Api from Django)

1

Crear un archivo **forms.py** (en la APP).

2

Importamos los forms.

3

Creamos una clase para crear un formulario.

The screenshot shows a code editor with a dark theme. On the left, the 'EXPLORER' sidebar shows a project structure for 'PROYECTOCODER' with a subdirectory 'AppCoder'. Inside 'AppCoder', files like '__pycache__', 'migrations', 'static', 'templates', '__init__.py', 'admin.py', 'apps.py', 'forms.py', and 'models.py' are listed. 'forms.py' is selected and highlighted. On the right, the editor shows the content of 'forms.py'. It starts with a comment 'AppCoder > forms.py > ...' followed by a line number 1 with the code 'from django import forms'. Line 2 is empty. Line 3 is empty. Line 4 starts a class definition 'class CursoFormulario(forms.Form):'. Line 5 is empty. Line 6 has a comment '#Especificar los campos'. Line 7 has 'curso = forms.CharField()'. Line 8 has 'camada = forms.IntegerField()'. Line 9 is empty. Line 10 is empty. Line 11 is empty.

```
EXPLORER
...
+ views.py M
+ forms.py U x
+ cursoFormulari

> OPEN EDITORS
✓ PROYECTOCODER
  ✓ AppCoder
    > __pycache__
    > migrations
    > static
    > templates
    + __init__.py
    + admin.py
    + apps.py
    + forms.py U
    + models.py

AppCoder > forms.py > ...
1  from django import forms
2
3
4  class CursoFormulario(forms.Form):
5
6      #Especificar los campos
7      curso = forms.CharField()
8      camada = forms.IntegerField()
9
10
11
```



Creación de formularios (Api from Django)

Script:

```
from django import forms
```

```
class CursoFormulario(forms.Form):
```

```
    curso = forms.CharField()
```

```
    camada = forms.IntegerField()
```




Creación de formularios (Api from Django)

4

Recibimos en el `html`, el formulario para crear el `<form>`

```
def cursoFormulario(request):  
    if request.method == 'POST':  
        miFormulario = CursoFormulario(request.POST) #aquí mellega toda la información del html  
        print(miFormulario)  
  
        if miFormulario.is_valid: #Si pasó la validación de Django  
            informacion = miFormulario.cleaned_data  
            curso = Curso (nombre=informacion['curso'], camada=informacion['camada'])  
            curso.save()  
            return render(request, "AppCoder/inicio.html") #Vuelvo al inicio o a donde quieran  
        else:  
            miFormulario= CursoFormulario() #Formulario vacio para construir el html  
    return render(request, "AppCoder/cursoFormulario.html", {"miFormulario":miFormulario})
```

```
from AppCoder.forms import CursoFormulario
```

```
def cursoFormulario(request):
```

```
    if request.method == "POST":
```

```
        miFormulario = CursoFormulario(request.POST) # Aqui me llega la informacion del html
```

```
        print(miFormulario)
```

```
        if miFormulario.is_valid:
```

```
            informacion = miFormulario.cleaned_data
```

```
            curso = Curso(nombre=informacion["curso"], camada=informacion["camada"])
```

```
            curso.save()
```

```
            return render(request, "AppCoder/inicio.html")
```

```
        else:
```

```
            miFormulario = CursoFormulario()
```

```
    return render(request, "AppCoder/cursoFormulario.html", {"miFormulario": miFormulario})
```

Script





Creación de formularios (Api from Django)

5

Modificamos la vista para que ya no reciba el **html**, sino el **api forms**.

```
from AppCoder.forms
import CursoFormulario.
```

Vemos que queda muy sencillo y cada vez nos desligamos más de html.

```
{% if miFormulario.errors %}

<p style="color: red;"> Están mal los datos, revisar</p>

{% endif %}

<form action="" method="POST">{% csrf_token %}

  <!-- Acá está la magia de Django-->

  <table>

    {{ miFormulario.as_table }}

  </table>

  <input type="submit", value="Enviar">

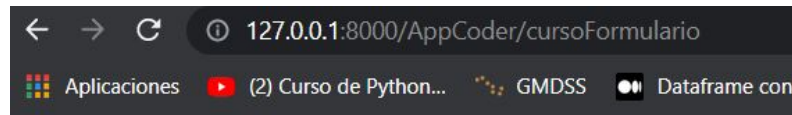
</form>
```

Creación de formularios (Api from Django)



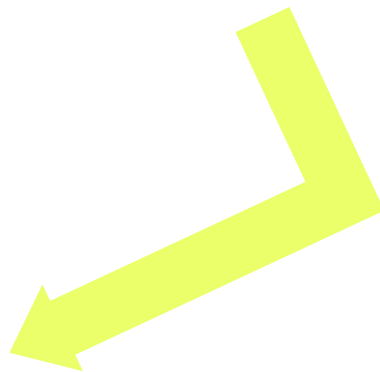
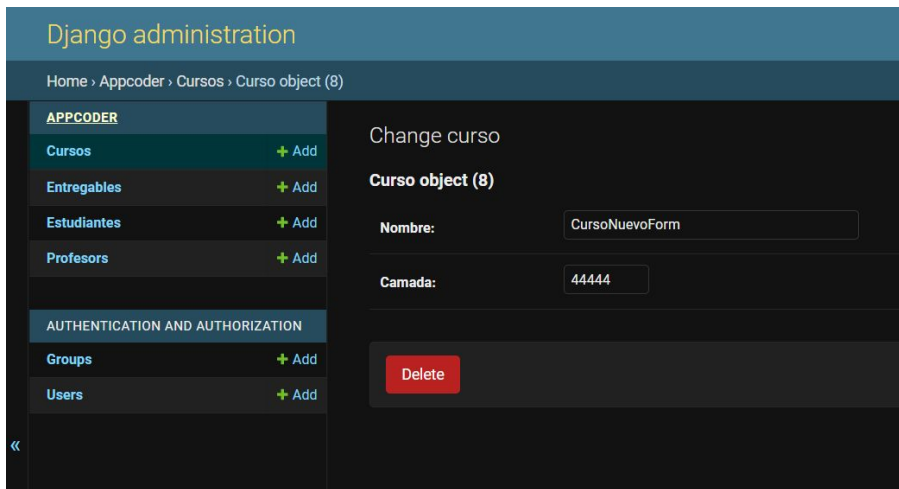
Curso:

Camada:



Curso:

Camada:





Ejemplo en vivo

Ahora carguemos algo que tenga más datos para ir perdiendo el temor. Con la práctica se hace el maestro 😊

Duración: **10 minutos**



Ejemplo en vivo

```
def profesorFormulario(request):  
    if request.method == 'POST':  
        miFormulario = ProfesorFormulario(request.POST) #aquí mellega toda la información del html  
        print(miFormulario)  
  
        if miFormulario.is_valid: #Si pasó la validación de Django  
            informacion = miFormulario.cleaned_data  
  
            profesor = Profesor (nombre=informacion['nombre'], apellido=informacion['apellido'],  
                                email=informacion['email'], profesion=informacion['profesion'])  
  
            profesor.save()  
  
            return render(request, "AppCoder/inicio.html") #Vuelvo al inicio o a donde quieran  
        else:  
            miFormulario= ProfesorFormulario() #Formulario vacio para construir el html  
  
    return render(request, "AppCoder/profesorFormulario.html", {"miFormulario":miFormulario})
```

Veamos la carga de nuevos datos

```
class ProfesorFormulario(forms.Form):  
    nombre= forms.CharField(max_length=30)  
    apellido= forms.CharField(max_length=30)  
    email= forms.EmailField()  
    profesion= forms.CharField(max_length=30)
```



Ejemplo en vivo

```
<h1>Formulario - Agregar Profesor</h1>

{% if miFormulario.errors %}

<p style="color: red;"> Están mal los datos, revisar</p>

{% endif %}

<form action="" method="POST">{% csrf_token %}

  <!-- Acá está la magia de Django-->

  <table>

    {{ miFormulario.as_table }}

  </table>

  <input type="submit", value="Enviar">

</form>
```

Estamos en la mitad del camino

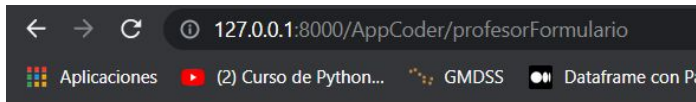
```
urlpatterns = [

    path('', views.inicio, name="Inicio"), #esta era nuestra primer view
    path('cursos', views.cursos, name="Cursos"),
    path('profesores', views.profesores, name="Profesores"),
    path('estudiantes', views.estudiantes, name="Estudiantes"),
    path('entregables', views.entregables, name="Entregables"),
    path('cursoFormulario', views.cursoFormulario, name="CursoFormulario"),
    path('profesorFormulario', views.profesorFormulario, name="ProfesorFormulario"),

]
```



Ejemplo en vivo



Formulario - Agregar Profesor

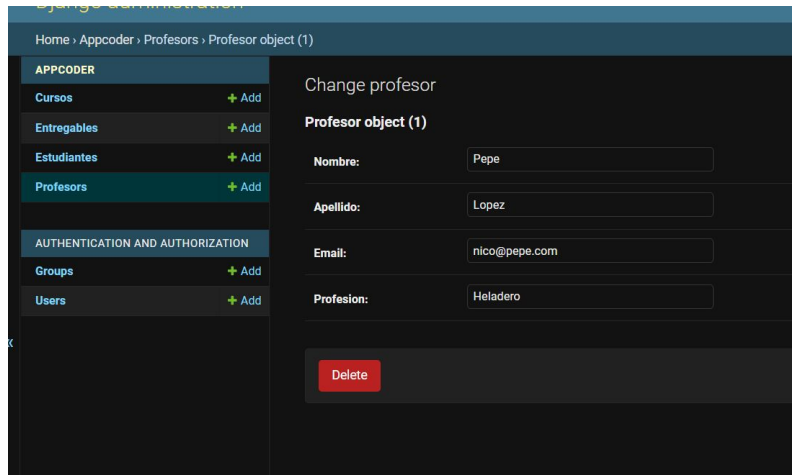
Nombre:

Apellido:

Email:

Profesion:

¡Increíble!
¿Sabrían que hemos conseguido?





Agregar con un API form

Agregar datos a una tabla de BD por medio de un Api Form.

Duración: **10 minutos**



ACTIVIDAD EN CLASE

Agregar con un API form

Descripción de la actividad.

Crear un Api Form que permita insertar datos a la base de datos de tu proyecto, una vez terminado verificar dichos datos en la BD.

Luego, instanciar a dos personas.



Break

¡10 minutos y volvemos!

Búsqueda con Form

Búsqueda con Form



También suele ser útil usar un formulario para buscar algún dato en nuestra base de datos.

Por ejemplo: si queremos saber si tenemos un curso que corresponda a una determinada camada, hay dos alternativas, o existe o no existe. En nuestro caso existe en la BD.

Curso object (5)

Nombre:	<input type="text" value="Desarrollo web"/>
Camada:	<input type="text" value="19881"/>



Búsqueda con Form

¡Ahora busquemos esa camada!

1. Creamos una vista, `busquedaCamada(request)`
2. Registramos el `url`
3. Creamos `busquedaCamada.html`
4. Creamos la vista buscar.

¡Veamos cómo realizarlo!



Búsqueda con Form

```
<form action="/AppCoder/buscar/" method="GET">

    <input type="number" name="camada" id="camada">

    <input type="submit" value="Buscar">

</form>
```

3

```
def busquedaCamada(request):

    return render(request, "AppCoder/busquedaCamada.html")

def buscar(request):

    respuesta = f"Estoy buscando la camada nro: {request.GET['camada'] }"

    #No olvidar from django.http import HttpResponse
    return HttpResponse(respuesta)
```

1

4

```
path('busquedaCamada', views.busquedaCamada, name="BusquedaCamada"),
path('buscar/', views.buscar),
```

2



Búsqueda con Form

← → ↻ ⓘ 127.0.0.1:8000/AppCoder/busquedaCamada

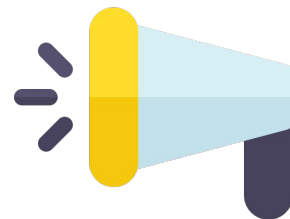
Aplicaciones (2) Curso de Python... GMDSS Dataframe

19881 | Buscar

← → ↻ ⓘ 127.0.0.1:8000/AppCoder/buscar/?camada=19881

Aplicaciones (2) Curso de Python... GMDSS Dataframe con Pan...

Estoy buscando la camada nro: 19881



**¡Anunciamos que lo
hemos logrado!**



Búsqueda con Form

Los datos llegaron, pero aún no buscamos en la BD.

Para ello, principalmente usaremos una búsqueda por filtros en la base de datos con **objects.filter**:

```
def buscar(request):  
  
    if request.GET["camada"]:  
  
        #respuesta = f"Estoy buscando la camada nro: {request.GET['camada'] }"  
        camada = request.GET['camada']  
        cursos = Curso.objects.filter(camada__icontains=camada)  
  
        return render(request, "AppCoder/resultadosBusqueda.html", {"cursos":cursos, "camada":camada})  
  
    else:  
  
        respuesta = "No enviaste datos"  
  
    #No olvidar from django.http import HttpResponse  
    return HttpResponse(respuesta)
```



Búsqueda con Form

Mientras que nuestro `resultadosPorBusqueda.html` quedaría así.

Notar que en nuestro caso teníamos más de un curso con esa camada, filter trae todos los registros, no solo uno.

```
<p>Estamos buscando el: {{camada}}</p>

{% if cursos %}

<ul>
  {% for curso in cursos %}
    <li> {{curso.nombre}} </li>
    <li> {{curso.camada}} </li>
  {% endfor %}
</ul>

{% endif %}
```

Ordenamiento de template

Ordenamiento de template

Con los formularios aprendimos a insertar nuevos datos y buscar datos en la base de datos. Próximamente aprenderemos a modificar datos y eliminar datos. Pero, hicimos muchos html y muy poco estéticos, ahora ordenaremos todo para tener **forms dentro html heredados de la planilla padre**.



Ordenamiento de template



InicioProfesoresCursosEstudiantesEntregables

APPCODER

Cursos + Add

Entregables + Add

Estudiantes + Add

Profesors + Add

AUTHENTICATION AND AUTHORIZATION

Groups + Add

Users + Add

Change curso

Curso object (9)

Nombre:Curso de costura

Camada:98989

Delete

Curso:Curso de costura

Camada:98989

Enviar



Ordenamiento de template

```
def cursos(request):  
  
    if request.method == 'POST':  
  
        miFormulario = CursoFormulario(request.POST) #aquí llega toda la información del html  
  
        print(miFormulario)  
  
        if miFormulario.is_valid: #Si pasó la validación de Django  
  
            informacion = miFormulario.cleaned_data  
  
            curso = Curso (nombre=informacion['curso'], camada=informacion['camada'])  
  
            curso.save()  
  
            return render(request, "AppCoder/inicio.html") #Vuelvo al inicio o a donde quieran  
  
    else:  
  
        miFormulario= CursoFormulario() #Formulario vacío para construir el html  
  
    return render(request, "AppCoder/cursos.html", {"miFormulario":miFormulario})
```

¿Cómo lo hice? 🙄.

¡Simple! Cambiando el url y la vista.

La vista que se llamaba **cursoFormulario**, pasó a llamarse **cursos** y me lleva a la página de **cursos.html** con un formulario.



Ordenamiento de template

```
urlpatterns = [  
    path('', views.inicio, name="Inicio"), #esta era nuestra primer view  
    path('cursos', views.cursos, name="Cursos"),  
    path('profesores', views.profesores, name="Profesores"),  
    path('estudiantes', views.estudiantes, name="Estudiantes"),  
    path('entregables', views.entregables, name="Entregables"),  
    #path('cursoFormulario', views.cursoFormulario, name="CursoFormulario"),  
    path('profesorFormulario', views.profesorFormulario, name="ProfesorFormulario"),  
    path('busquedaCamada', views.busquedaCamada, name="BusquedaCamada"),  
    path('buscar/', views.buscar),  
]
```



Ordenamiento de template

```
{% extends "AppCoder/padre.html" %}

{% load static %}

{% block contenidoQueCambia %}
    <!--Aquí va lo que cambia, y lo asociado a está vista :)-->
    {% if miFormulario.errors %}

        <p style="color: red;"> Están mal los datos, revisar</p>

    {% endif %}

    <form action="" method="POST">{% csrf_token %}

        <!-- Acá está la magia de Django-->

        <table>
            {{ miFormulario.as_table }}
        </table>

        <input type="submit", value="Enviar">

    </form>

{% endblock %}
```




Para pensar

¿Cómo harías para usar el mismo html para buscar y para ver los resultados de la búsqueda? ¿Se podrá?

Contesta mediante el chat de Zoom

Respuesta

¡Claro que sí!

Solo resta hacer que la misma plantilla resuelva los POST afirmativos, nulos o incorrectos, así se podría modificar.

```
{% if cursos %}

<p>Estamos buscando el: {{camada}}</p>

<ul>
{% for curso in cursos %}
    <li> {{curso.nombre}} </li>
    <li> {{curso.camada}} </li>
{% endfor %}
</ul>

{% else %}
<p>No hay datos con esa descripción.</p>

{% endif %}

<p style="color: red;">{{respuesta}}</p>
```

```
def buscar(request):

    if request.GET["camada"]:

        #respuesta = f"Estoy buscando la camada nro: {request.GET['camada']}"
        camada = request.GET['camada']
        cursos = Curso.objects.filter(camada__icontains=camada)

        return render(request, "AppCoder/inicio.html", {"cursos":cursos, "camada":camada})

    else:

        respuesta = "No enviaste datos"

        #No olvidar from django.http import HttpResponseRedirect
        #return HttpResponseRedirect(respuesta)
        return render(request, "AppCoder/inicio.html", {"respuesta":respuesta})
```



Tercera pre-entrega de tu Proyecto final

Deberás entregar **la tercera pre-entrega**, correspondiente a tu proyecto final.



Tercera entrega

Objetivos generales

- ✓ Desarrollar una WEB Django con patrón MVT subida a Github.

Se debe entregar

- ✓ Link de GitHub con el proyecto totalmente subido a la plataforma.
- ✓ Proyecto Web Django con patrón MVT que incluya:
 1. Herencia de HTML.
 2. Por lo menos 3 clases en models.
 3. Un formulario para insertar datos a todas las clases de tu models.
 4. Un formulario para buscar algo en la BD
 5. Readme que indique el orden en el que se prueban las cosas y/o donde están las funcionalidades.



Tercera entrega

Formato

- ✓ Link al repositorio de GitHub con el nombre “**Tercera pre-entrega+Apellido**”.

Sugerencias

- ✓ Activar comentarios en el archivo y usar como guía el proyecto subido al material complementario de esta clase. También pueden obtener la rama de Git que tiene el mismo material [Rama-De-Git](#).

Recuerden que en la clase 1 tienen disponible un proyecto modelo de un ex estudiante para tomar como inspiración.

¿Preguntas?

Resumen de la clase hoy

- ✓ Formularios de alta de BD.
- ✓ Formularios de búsqueda.
- ✓ Formularios en htmls heredados.

Opina y valora
esta clase

Muchas gracias.

#DemocratizandoLaEducación