



**¡Les damos la
bienvenida!**

¿Comenzamos?

Esta clase va a ser

- grabada
a

Semana 10. PYTHON

Portfolio y Playground Intermedio parte I

Objetivos de la clase

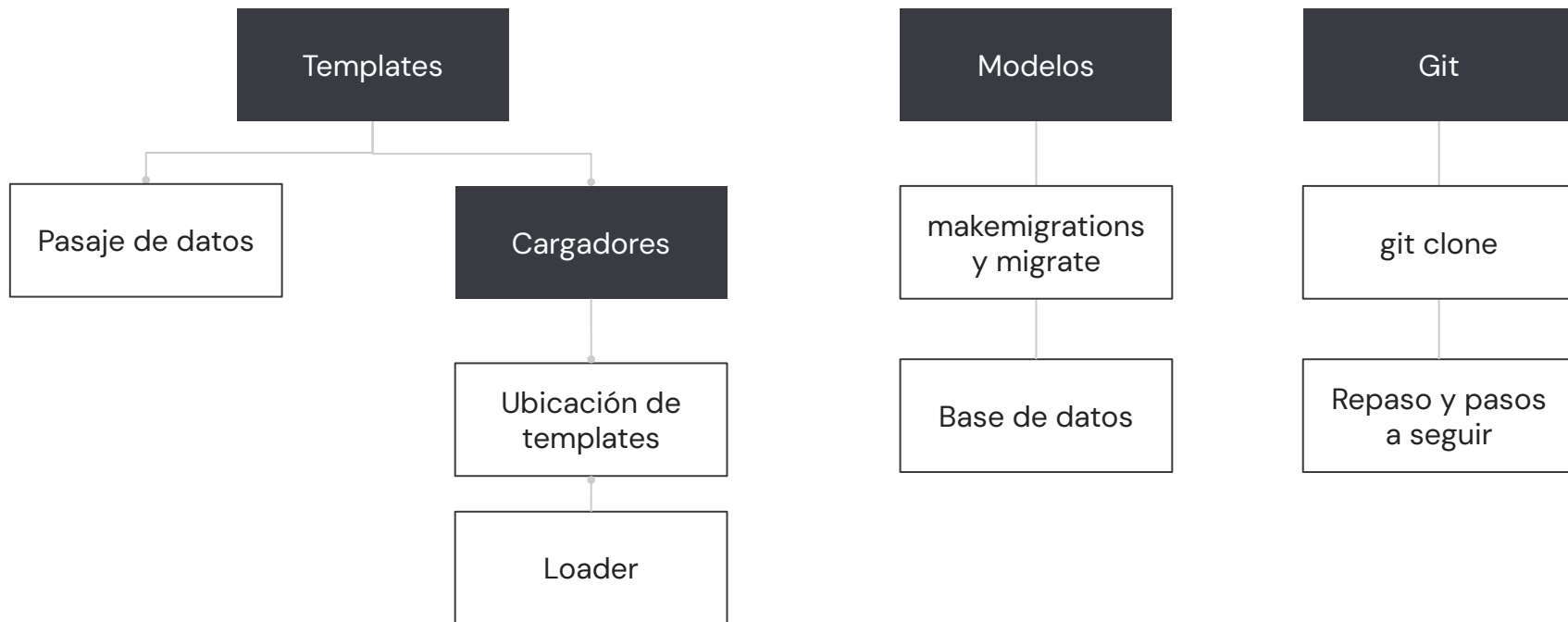
- **Crear** nuestro portfolio.
- **Profundizar** conceptos de MTV.
- **Ejecutar** cambios en nuestro Git y **gestionar** las versiones en GitHub.
- **Crear** URLs avanzadas.



¡Recuerda esto!

Antes de iniciar esta clase,
debes abrir VSC y DB Browser for
SQLite (opcional)

MAPA DE CONCEPTOS



Repasemos...



REPASO

Semana 10

En los videos de esta semana aprendiste sobre:

- ✓ Pasaje de datos a templates
- ✓ Cargadores
- ✓ Creación de modelos
- ✓ Generar y ejecutar migraciones
- ✓ Seguimos los pasos necesarios para tener nuestro proyecto subido y cómo manejar los cambios.



VIDEO N°10.1 – MEJORAS EN LAS
PLANTILLAS

Recuperamos el tema visto

Agregamos el **pasaje de información desde vistas por medio del contexto a los templates/plantillas**. Esto no solo permite pasar valores individuales, sino que también podemos pasar colección de datos como listas, tuplas, etc. Hay que tener en cuenta que al contexto se le debe pasar la información estructurada en un diccionario.

No solo nos quedamos con eso, sino que mejoramos la implementación de los templates con el uso de cargadores para simplificar el código de la vista, obteniendo el mismo resultado.





VIDEO N° 10.2 – MODELO

Recuperamos el tema visto

Para **crear un modelo** se debe hacer dentro del archivo `models.py` de la app relacionada, heredando de `models.Model` y definiendo los tipos de campos necesarios para cada atributo de la clase que queremos se guarde en la base de datos (bd).

Para que el proyecto sepa que se crearon los modelos de una app debemos informarle que la app está dentro de las apps instaladas en el `settings.py` y también ejecutar los comandos `makemigrations` para que se genere el código que configurara la bd y luego el comando `migrate` que ejecuta el código generado por el comando anterior y genera los cambios requeridos en la bd.





VIDEO N°10.3 – PROFUNDIZANDO MVT

Recuperamos el tema visto

Pudimos ver cómo **subir nuestro proyecto a GitHub** cambiando algunos pasos por medio del comando Git clone.

El comando git clone no solo sirve en el momento de iniciar un proyecto, sino que es el comando utilizado para bajarnos proyectos que estén subidos y queramos tenerlos, ya sea porque estamos en otra computadora o nos lo están compartiendo para que lo usemos.



CODERHOUSE

Contenido pregrabado

Si en algún momento quieres buscar un fragmento de contenido pregrabado y no recuerdas dónde hacerlo, puedes consultar [este resumen](#). Luego, podrás ir directo a buscar el video o podcast que necesites.

¿Preguntas?

Te invitamos a dejar tu
pregunta a través de/del
[chat](#)



Puesta en común microdesafío

¡Vamos a recuperar lo trabajado durante la semana!

Duración: **10 minutos.**



PUESTA EN COMÚN – MICRODESAFÍO

Crea tu modelo

Crea un modelo, el cual deberá tener por lo menos 2 atributos y ejecutar los comandos necesarios para que la base de datos los reconozca.



RESULTADOS – MICRODESAFÍO

Crea tu modelo

¿Qué aprendimos?

Creación de modelos, definición de atributos, uso de migraciones.

¡Buen trabajo! 🧐



Para pensar

¿Cómo lograrían esto?



Contesta mediante el chat de Zoom



Agregando cargadores

Modifica alguna de tus vistas para que funcione usando cargadores.

Duración: **10 min**



ACTIVIDAD

Agregando cargadores

Descripción de la actividad.

Agregar la url, vista y template para el modelo del microdesafío. Para la vista y carga de template utiliza cargadores.





Puesta en común

Duración: 5 min

Agregar información a mi BD



Ejemplo en vivo: Carga en BD

Agregaremos info a nuestra BD desde consola y desde DB Browser.

Duración: **15 minutos**



Agregar información a mi BB.DD

- ✓ Agregaremos algún registro, por ejemplo agregaremos algún curso.
- ✓ Vamos a la consola y escribimos `python manage.py shell` para pararnos en la consola.
- ✓ Luego vamos a importar el modelo, en este caso la clase Curso.
 - `from AppCoder.models import Curso`
- ✓ Pasamos a crear un Curso:
`curso = Curso(nombre="Python", camada=23800)`
- ✓ Luego enviamos ese curso a la BD: `curso.save()`

The screenshot shows a database management interface with a menu bar (File, Edit, View, Tools, Help) and buttons for 'New Database', 'Open Database', and 'Write'. Below these are tabs for 'Database Structure', 'Browse Data', and 'Edit Pragmas'. The 'Table:' dropdown is set to 'AppCoder_curso'. The table structure is shown with columns 'id', 'nombre', and 'camada'. The first row contains the values '1', 'Python', and '23800'.

	id	nombre	camada
	Filter	Filter	Filter
1	1	Python	23800

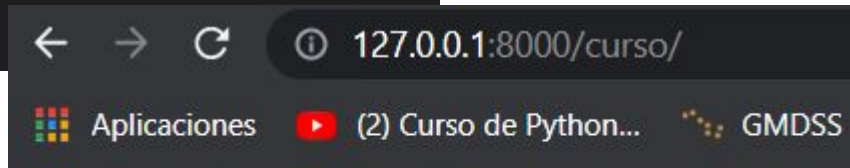


Agregar información a mi BD

Ya probamos desde la consola que todo funciona, ahora miremos cómo quedaría una vista que guarda datos y luego los muestra en la web. Sería lo mismo, pero solo haciendo una vista, y modificando el archivo urls.py.

```
def curso(self):  
  
    curso = Curso(nombre="Desarrollo web", camada="19881")  
    curso.save()  
    documentoDeTexto = f"--->Curso: {curso.nombre}    Camada: {curso.camada}"  
  
    return HttpResponse(documentoDeTexto)
```

```
from AppCoder.views import curso  
  
urlpatterns = [  
    path('admin/', admin.site.urls),  
    path('curso/', curso)  
]
```



--->Curso: Desarrollo web Camada: 19881



Otra manera de agregar información a mi BD

DB Browser for SQLite - C:\Users\layla\Escritorio\ProyectoCoder\ProyectoCoder\db.sqlite3

File Edit View Tools Help


New Database Open Database Write Changes Revert Changes Open Project Save Project Attach Database Close Database

Database Structure Browse Data Edit Pragmas Execute SQL

Table: AppCoder_curso

Filter in any column

	id	nombre	camada
	Filter	Filter	Filter
1	1	Python	23800





Otra manera de agregar información a mi BD

Agreguemos ahora la camada del curso de JavaScript. De la siguiente manera:

The screenshot shows the DB Browser for SQLite interface. The main window displays a table named 'AppCoder_curso' with three columns: 'id', 'nombre', and 'camada'. The table contains two rows. The second row, with 'id' 2 and 'nombre' 2, is selected. A red arrow points to this row. The 'Edit Database Cell' dialog is open, showing the value 'JavaScript' in the text field. The dialog also shows the data type as 'Text / Numeric' and the character limit as '10 character(s)'. The 'Apply' button is visible.

id	nombre	camada
1	Python	23800
2	2	0



Otra manera de agregar información a mi BD

Hacemos el mismo procedimiento pero ahora con el nro. de la Camada:

The screenshot shows the DB Browser for SQLite interface. The main window displays a table named 'camada' with columns 'id', 'nombre', and 'camada'. The table contains two rows: (1, 'Python', 23800) and (2, 'JavaScr...', 0). A red arrow points to the cell containing '0' in the 'camada' column for row 2. An 'Edit Database Cell' dialog is open on the right, showing the current value '23456' in a text input field, which is highlighted with a red box. The dialog also shows the data type as 'Text' and the character count as '5 character(s)'.

id	nombre	camada
1	1 Python	23800
2	2 JavaScr...	0

1 23456

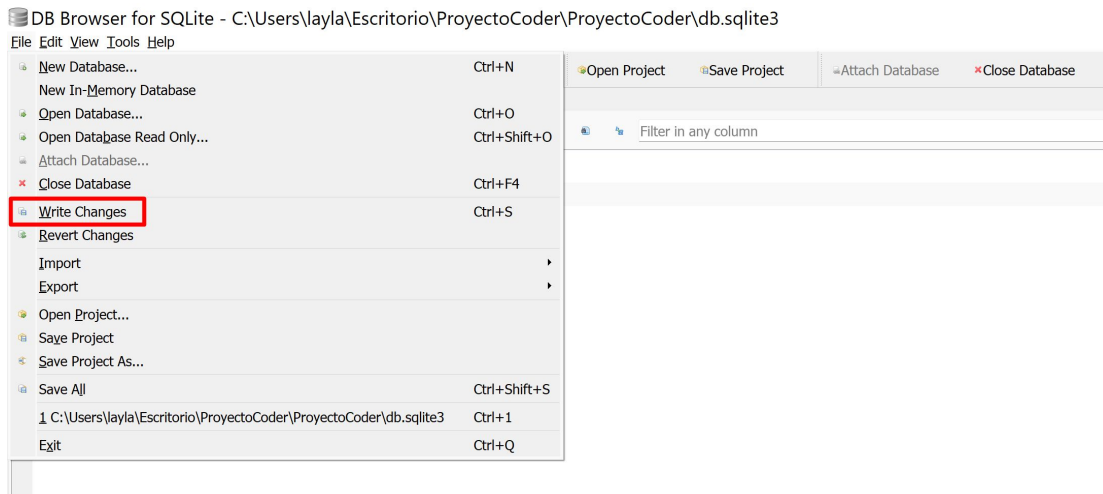
Type of data currently in cell: Text / Numeric
5 character(s)

Apply



Otra manera de agregar información a mi BD

Hacemos persistir los cambios 😊





Break

¡En 10 minutos volvemos!



Mi Django a Github

Enviarás una versión de tu proyecto a GitHub.

Duración: **15 min**



ACTIVIDAD

Mi Django a Github

Descripción de la actividad.

Trabajarás sobre un Proyecto Web de Django que ya tengas funcionando y lo subirás a un Repositorio de GitHub, crear dicho repositorio en caso de no tenerlo.

Recuerda o consulta el paso a paso de la clase para poder ejecutarlo.

Vistas y URLs (URLs avanzadas)

Creando vistas



Estamos tratando de **estructurar nuestro proyecto para incorporar nuevos conceptos.**

Iniciemos **creando algunas vistas** asociadas a nuestro modelo:

```
def inicio(request):  
    return HttpResponseRedirect('vista inicio')  
  
def cursos(request):  
    return HttpResponseRedirect('vista cursos')  
  
def profesores(request):  
    return HttpResponseRedirect('vista profesores')  
  
def estudiantes(request):  
    return HttpResponseRedirect('vista estudiantes')  
  
def entregables(request):  
    return HttpResponseRedirect('vista entregables')
```

URLs

Organicemos nuestras URLs

Un proyecto puede tener muchas App (nosotros tenemos solo una), pero pensando en algo más general, acomodemos un poco las urls, para mejorar la reutilización.



Para pensar

¿Qué pasaría si nuestro archivo `urls.py` tuviera que dirigir a MUCHAS APPS? ¿Qué alternativa se te ocurre?

Contesta mediante el chat de Zoom

**Generar archivo URL. PY
en nuestra app**

Paso a paso

1. Crear en AppCoder urls.py
2. Le importamos el path: `from django.urls import path`
3. Importamos las vistas `from AppCoder import views`
4. Copiamos y pegamos el urlpatterns que ya teníamos, pero sin el admin
5. Dejamos el admin, SOLO, en el url del Proyecto (tampoco necesita las vistas)
6. Lo más importante, relacionamos el urls.py de la App con el Proyecto:
7. `path('AppCoder/', include('AppCoder.urls'))`



Ejemplo en vivo

Veamos cómo generar URL.PY

Duración: **5 minutos**

URLs



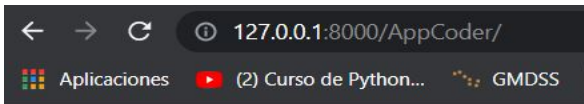
URLs del Proyecto

```
from django.contrib import admin
from django.urls import path, include
#from AppCoder.views import * #Ya no seria necesario :)

urlpatterns = [
    path('admin/', admin.site.urls),

    path('AppCoder/', include('AppCoder.urls')),
]
```

Todo está bien 😊



vista inicio

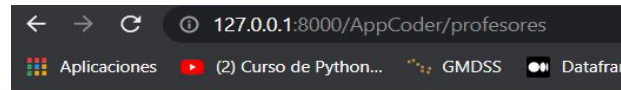
URLs de la App

```
from django.urls import path

from AppCoder import views

urlpatterns = [

    path('', views.inicio), #esta era nuestra primer view
    path('cursos', views.cursos, name="Cursos"),
    path('profesores', views.profesores),
    path('estudiantes', views.estudiantes),
    path('entregables', views.entregables),
]
```



vista profesores

Creando nuestros Templates

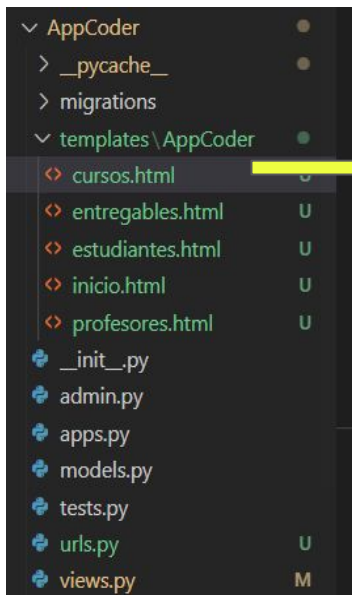


Ejemplo en vivo

Veamos cómo pulir nuestro Template.

Duración: **10 minutos**

Puliendo estructura y definiciones



1- Vamos a la App y creamos una carpeta templates, y dentro de ella una subcarpeta que se llame AppCoder

```
AppCoder > templates > AppCoder > cursos.html > html
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>Cursos</title>
8  </head>
9  <body style="background-color: blue;">
10
11 </body>
12 </html>
```

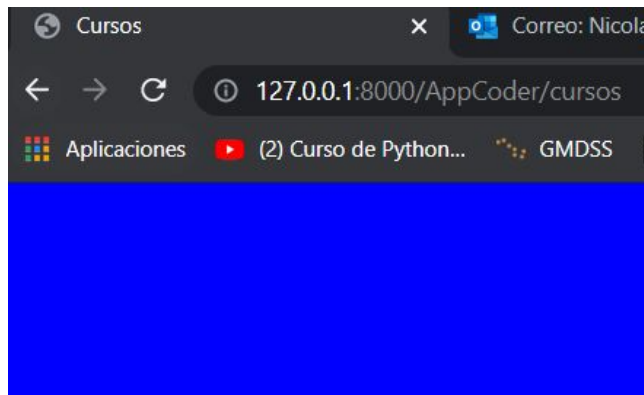
2- Dentro de esta última creamos los html. Uno por vista.

Puliendo estructura y definiciones



```
def inicio(request):  
    return render(request, "AppCoder/inicio.html")  
  
def cursos(request):  
    return render(request, "AppCoder/cursos.html")  
  
def profesores(request):  
    return render(request, "AppCoder/profesores.html")  
  
def estudiantes(request):  
    return render(request, "AppCoder/estudiantes.html")  
  
def entregables(request):  
    return render(request, "AppCoder/entregables.html")
```

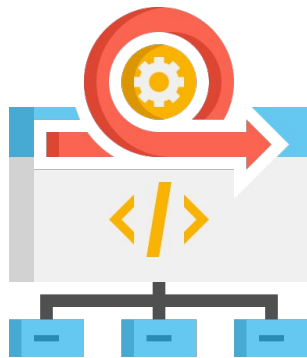
3- Hacemos que las vistas hagan Render de los templates:



Mejorando nuestros Templates

Mejorando nuestros Templates

Si bien no ahondaremos sobre desarrollo web, necesitamos archivos **.html**. Para simplificar ésto usaremos otro Framework (**como Django**). Esta nueva herramienta nos ayudará a que nuestra web sea más “linda” sin saber mucho.



Mejorando nuestros Templates

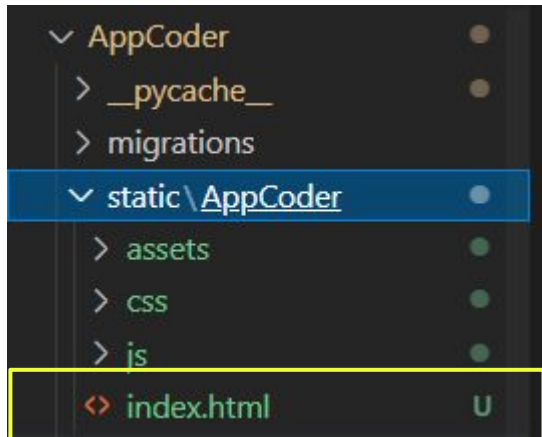


1- Haremos templates con Bootstrap.
Vamos a <https://getbootstrap.com/> para saber más, y nos descargamos algún esqueleto cualquiera.



2- En nuestros ejemplos usaré:
<https://startbootstrap.com/previews/landing-page>

Mejorando nuestros Templates



3- Creamos una carpeta en nuestra App, debe llamarse static, dentro de ella AppCoder. Dentro de esta última pondremos el esqueleto que nos bajamos.

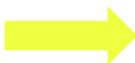
Para eso copiamos todo el contenido en inicio.html

Este será nuestro nuevo inicio.html

Mejorando nuestros Templates



4- No queremos que se vea así,
¿Verdad?:



☐

5 - Para evitar eso, aplicamos lo
siguiente:

a- Agregar los archivos de static:

```
<head>  
  {% load static %}  
  
  <meta charset="utf-8" />  
  <meta name="viewport" content="width=device-width, initial-scale=1">  
  <meta name="description" content="Coderhouse - The best place to learn web development in Spanish.">  
  <meta name="author" content="Coderhouse" />
```

b- Cambiamos direcciones

Fully Responsive

This theme will look great on any device, no matter the size!



Bootstrap 5 Ready

Featuring the latest build of the new Bootstrap 5 framework!



Easy to Use

Ready to use with your own content, or customize the source files!



Mejorando nuestros Templates

Antes 😞

```
<!-- Core theme CSS (includes Bootstrap)-->  
<link href="css/styles.css" rel="stylesheet" />
```

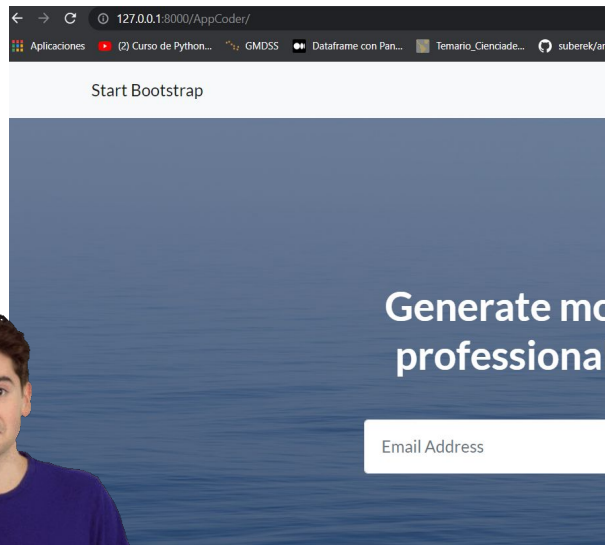
Después 😊

```
<!-- Core theme CSS (includes Bootstrap)-->  
<link href="{% static 'AppCoder/css/styles.css' %}" rel="stylesheet" />
```

Mejorando nuestros Templates



¡Listo! Tenemos una web estéticamente correcta que iremos completando con datos y lógica que proviene de Django.



Último retoque al Template

Mejorando nuestros Templates



Dejaremos la Web con este aspecto, para tener un menú inicial, un pie de página y un cuerpo.

¿Cómo? Veámoslo en el vivo...





Ejemplo en vivo

Vamos a modificar el template por defecto para solo dejar lo que nos interesa sin perder la estética y sin saber nada de desarrollo web.

Duración: **15 minutos**

¿Preguntas?

Te invitamos a dejar tu
pregunta a través de/del
[chat](#)

¡Para no olvidarse!

La próxima semana se encuentra pautada la **Pre-entrega 3: Tu primera página**. Una vez visto el contenido pregrabado de dicha semana, les recomendamos empezar a trabajar en ella.





MATERIAL AMPLIADO

- ✓ [Agregar información a la BD desde consola y desde DB Browser](#) | Nicolás Pérez
- ✓ [Instalar BootStrap 5 en tu proyecto](#) | Kiko Palomares

¿Preguntas?

Resumen de la clase hoy

- ✓ Python en HTML
- ✓ Definimos y creamos nuestro primer Modelo
- ✓ Diferenciamos entre proyecto y app.
- ✓ Mejoramos la relación Template - URLs
- ✓ Construimos URLs en nuestra APP
- ✓ Creamos templates más profesionales

La próxima semana

Los próximos temas que vamos a ver



Contenido Pregrabado

- ✓ Video 11.1 – Herencia de templates
- ✓ Video 11.2 – Panel de administración
- ✓ Video 11.3 – Formularios
- ✓ Microdesafío – Heredemos



Clase en vivo (2h)

- ✓ Ejemplo en vivo: Instancias
- ✓ Actividad individual: Agregar datos a nuestro model
- ✓ Ejemplo en vivo: Formularios
- ✓ Actividad individual: Agregar con un API form
- ✓ Actividad individual: Completemos



Instancia de evaluación

- ✓ Pre-entrega N° 3 – Tu primera página

La **próxima** semana

Recuerda que, a partir de ahora, tienes disponible el contenido pregrabado en la plataforma y que **es necesario que lo veas en forma previa a la próxima clase.**

Opina y valora
esta clase

Muchas gracias.

#DemocratizandoLaEducación

**¡Gracias por estudiar
con nosotros! ✨**