



PROGRAMACION III

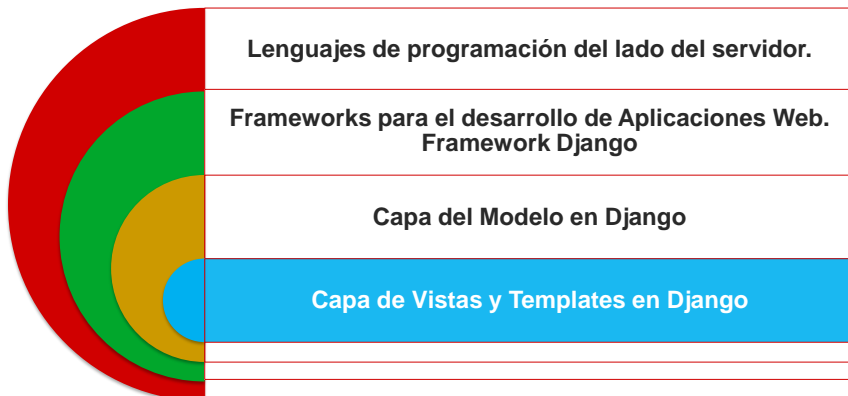
UNIDAD IV: DJANGO – VISTAS Y TEMPLATES

DOCENTES DE CATEDRA:

- ❖ Mgtr. Cecilia E. Gallardo
- ❖ Esp. Marta Miranda



UNIDAD IV: PROGRAMACIÓN DEL LADO DEL SERVIDOR





Django. Primer Proyecto Web

Vistas (Views) basadas en funciones

- Una **vista basada en función** de Django es una función de Python que recibe una solicitud HTTP y devuelve una respuesta HTTP. Se incluye aquí toda la lógica necesaria para devolver la respuesta deseada.
- La secuencia de actividades para convocar una vista es la siguiente:
 - 1) Crear las vistas de la aplicación
 - 2) Definir un patrón de URL para cada vista (archivo urls.py)
 - 3) Crear plantillas (templates) HTML para representar los datos generados por las vistas. Cada vista renderizará una plantilla, le pasará variables y devolverá una respuesta HTTP con la salida renderizada.
- Las vistas también pueden manejadas como **vistas basadas en clases**. Estas se implementan como objetos de Python en lugar de funciones, heredando de la clase **View**, que maneja el envío de métodos HTTP y otras funcionalidades comunes.

<https://docs.djangoproject.com/en/3.1/topics/class-based-views/intro/>

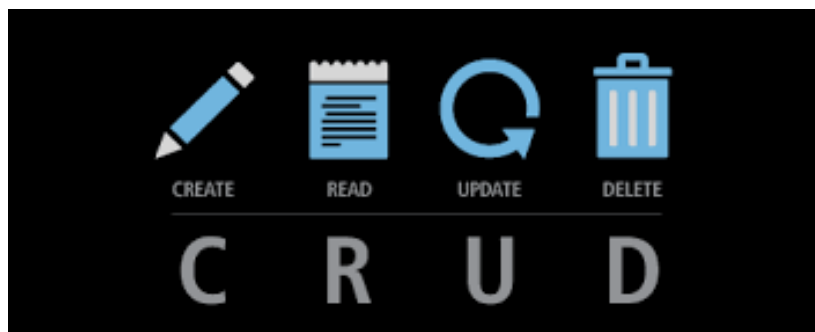
[Programación III]

[3]



Django. Primer Proyecto Web

Ejemplos de operaciones CRUD en Vistas y Templates



[Programación III]

[4]

Django. Primer Proyecto Web

Creación de vista que lista los Programas del caso de estudio

- Editar el archivo `views.py` de la aplicación `programa`:

```
views.py x
1 from django.shortcuts import render, get_object_or_404
2 from .models import Programa
3
4 def programa_lista(request):
5     programas = Programa.objects.all()
6     return render(request, 'programa/lista.html',
7                   {'programas': programas})
```

- La vista `programa_lista` recibe el objeto `request` como único parámetro, el cual es obligatorio para todas las vistas.
- En esta vista, se recuperan todos los programas.
- Luego mediante la función `render()` se renderiza la lista con la plantilla dada, a la cual se le pasan los parámetros: objeto `request`, ruta de la plantilla y variables de contexto. Esta función finalmente devuelve un objeto `HttpResponse` con el texto renderizado (normalmente código HTML).

[Programación III]

[5]

Django. Primer Proyecto Web

Creación de vista que muestra el detalle de un Programa

- Ahora crearemos una segunda vista para mostrar el detalle de un Programa:

```
def programa_detalle(request, pk):
    programa = get_object_or_404(Programa, pk=pk)
    return render(request,
                  'programa/detalle_programa.html',
                  {'programa': programa})
```

- Esta vista toma como argumento la `pk` (primary key) o `id` de un programa, para recuperar uno en particular.
- Se utiliza aquí la función `get_object_or_404()` que recupera un objeto que coincida con los parámetros dados o, lanza una excepción `HTTP 404` (no encontrado) si no se encuentra ningún objeto.
- Finalmente, la vista utiliza la función `render()` para renderizar el programa recuperado usando una plantilla.

[Programación III]

[6]



Django. Primer Proyecto Web

Creación de vistas de lista y detalle de Programa

El objeto **REQUEST** de la clase **HttpRequest**, contiene entre otros, los siguientes atributos: (<https://docs.djangoproject.com/en/3.1/ref/request-response/>)

- **path**: un String que representa la ruta completa a la página solicitada, sin incluir el esquema o el dominio. Ejemplo: `"/musica/bandas/the_beatles/"`
- **method**: String que representa el método HTTP utilizado en la solicitud. Valores posibles: GET / POST
- **GET / POST**: instancia de **django.http.QueryDict**, de tipo diccionario que contiene todos los parámetros HTTP GET/POST adjuntados y que provienen de formularios HTML (POST) o de la URL (GET).
- **FILES**: un objeto de tipo diccionario que contiene todos los archivos cargados mediante la etiqueta `<input type = "file" name = "xx">` dentro de un formulario `<form enctype="multipart/form-data">`
- **session**: un objeto tipo diccionario que representa la sesión actual
- **user**: una instancia de **AUTH_USER_MODEL** que representa al usuario actualmente logueado. Si el usuario no ha iniciado sesión actualmente, el usuario devuelto será una instancia de *AnonymousUser*.

[Programación III]

[7]



Django. Primer Proyecto Web

Agregar patrones de URL para las vistas creadas

- Los **patrones de URL** permiten asignar **URLs** a **vistas**.
- Los mismos se componen de: *un patrón String URL*, *una vista* y, opcionalmente, *un nombre* que permite nombrar la URL en todo el proyecto.
- Dada una petición de usuario, Django recorre cada patrón URL y se detiene en el primero que coincide con la URL solicitada.
- Luego, Django importa la vista del patrón URL coincidente y la ejecuta, pasando una instancia de la clase **HttpRequest** y argumentos posicionales o de palabra clave (diccionarios).
- Crear un archivo **urls.py** en el directorio de la aplicación programa:

```
1 from django.urls import path
2 from . import views
3
4 app_name = 'programa'
5
6 urlpatterns = [
7     # programa views
8     path('', views.programa_lista, name='programa_lista'),
9     path('<int:pk>/', views.programa_detalle,
10         name='programa_detalle'),
11 ]
```

[Programación III]

[8]



Django. Primer Proyecto Web

Agregar patrones de URL para las vistas creadas

- En el código anterior, se define un **espacio de nombres de aplicación** con la variable **app_name**.
- Esto permite organizar las URL por aplicación y usar ese nombre al referirse a ellas.
- Se definieron dos patrones diferentes usando la función **path()**:
 - 1) El primer patrón URL no recibe argumentos y se asigna a la vista **programa_lista**:
`path('', views.programa_lista, name='programa_lista')`
 - 1) El segundo patrón recibe un argumento denominado **pk** que requiere un tipo entero y se asigna a la vista **programa_detalle**:
`path('<int:pk>/', views.programa_detalle, ...)`
- Se utilizan corchetes angulares <> para capturar los valores de variables de la URL, y los mismos se reciben como un *String*.
- Por ello, se deben utilizar convertidores de ruta, como **<int:pk>**, para hacer coincidir y devolver específicamente un entero.

[Programación III]

[9]



Django. Primer Proyecto Web

Agregar patrones de URL de aplicación al proyecto

- A continuación, se deben incluir los patrones de URL de la aplicación **programa** en los patrones de URL del proyecto.
- Editar el archivo **urls.py** ubicado en el directorio **asistencias** del proyecto:

```

16 from django.contrib import admin
17 from django.urls import path, include
18
19 urlpatterns = [
20     path('admin/', admin.site.urls),
21     path('programa/', include('apps.programa.urls', namespace='programa')),
22 ]

```

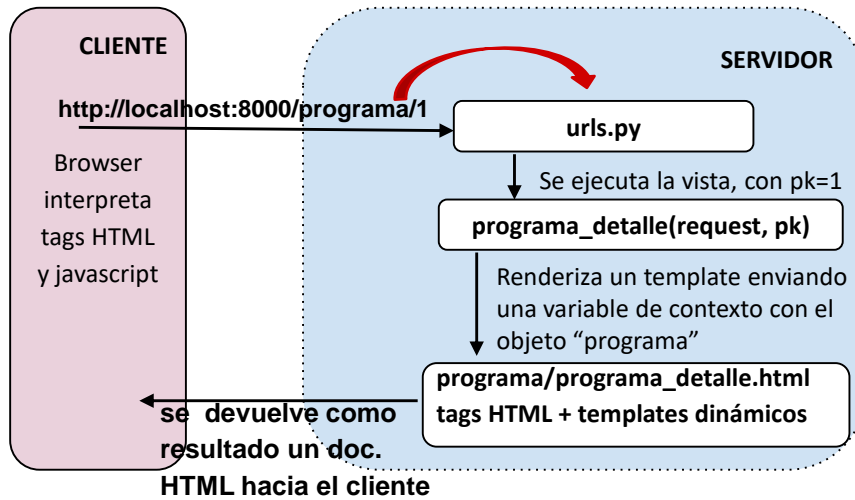
- El patrón URL definido con **include** se refiere a las URLs definidas en la aplicación de programa que se incluyan bajo la ruta **"programa/"**
- Al definir un espacio de nombre por aplicación, se podrá consultar las URL como sigue por ej: **programa:programa_lista** y **programa:programa_detalle**
- Ver mas en: <https://docs.djangoproject.com/en/3.0/topics/http/urls/#url-namespaces>

[Programación III]

[10]

Django. Primer Proyecto Web

Ejemplo de funcionamiento de vista "detalle_programa"



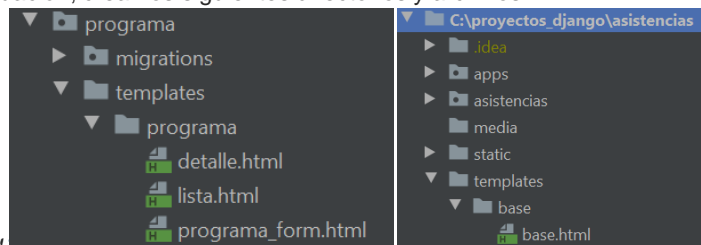
[Programación III]

[11]

Django. Primer Proyecto Web

Creación de plantillas o templates

- Hasta este punto, se han creado vistas y patrones URL para la aplicación **programa**.
- Los patrones URL relacionan las URL con las vistas y las vistas deciden QUE datos se devuelven al usuario.
- Las **plantillas** definen COMO se muestran los datos; generalmente están escritas en HTML en combinación con el lenguaje de plantilla Django.
- Ver más en: <https://docs.djangoproject.com/en/3.1/ref/templates/language/>
- A continuación, crear los siguientes directorios y archivos HTML:



[Programación III]


[12]



Django. Primer Proyecto Web

Creación de plantillas o templates

- Una configuración importante que se debe realizar es indicar la ruta predeterminada para recuperar las plantillas de todo el proyecto.
- Para ello, en el archivo **settings.py** agregar el siguiente elemento a la configuración de 'DIRS' de la variable **TEMPLATES**:



```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {...},
    },
]
```

La función **join** del modulo de Python **os.path**, une uno o más componentes de ruta de forma inteligente.

[Programación III]

[13]



Django. Primer Proyecto Web

Creación de plantillas o templates

- El archivo **base.html** incluirá el layout principal HTML de la aplicación web que se repite en todas las paginas web (encabezados, menús, pie de pagina, etc.).
- Los archivos **lista.html** y **detalle.html** heredarán del archivo **base.html** para representar la lista de programas y la vista detallada, respectivamente.
- Django proporciona un sistema de plantillas que permite especificar cómo se muestran los datos. Se basa principalmente en **bloques**, **etiquetas**, **variables** y **filtros de plantilla**.
- Los **bloques** definen una sección que puede ser reemplazada por plantillas que heredan de una base. Sintaxis: `{% block nombre %} {% endblock %}`
- Las **etiquetas de plantilla** controlan la representación y lógica de la plantilla. Sintaxis: `{% tag %}`. Ejemplos: `{% if %}` `{% for %}` `{% url %}` **etc.**
- Las **variables de plantilla** se reemplazan con valores cuando la plantilla se renderiza. Sintaxis: `{{variable}}`
- Los **filtros de plantilla** permiten hacer modificaciones al valor de las variables que se muestran. Sintaxis: `{{variable | filtro}}`

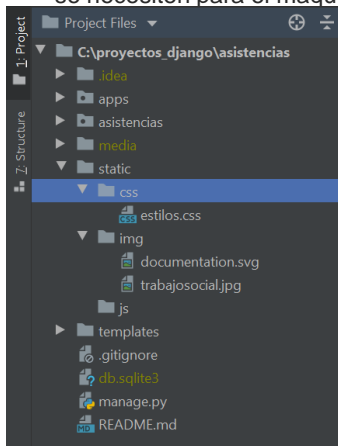
[Programación III]

[14]

Django. Primer Proyecto Web

Creación de plantillas o templates – Archivos estáticos

- Dentro del directorio principal del proyecto, crear una carpeta “**static**” para colocar allí todos los archivos de hojas de estilos, imágenes y javascript que se necesiten para el maquetado de la aplicación web.



- Luego, en el archivo “settings.py” configurar la url de los archivos estáticos:

```
STATIC_URL = '/static/'
STATICFILES_DIRS = (os.path.join(BASE_DIR, 'static'),)
```

[Programación III]

[15]

Django. Primer Proyecto Web

Creación de plantillas o templates – Archivo: base.html

```
1 {% load static %}
2 <!DOCTYPE html>
3 <html>
4   <head>
5     <title>{% block titulo %}{% endblock %}</title>
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <link href="{% static 'css/estilos.css' %}" rel="stylesheet">
8   </head>
9   <body>
10     <div class="contenedor">
11       <header>
12         <a href="/"> </a>
14         <h1>Sistema de Gestión de Asistencias Sociales</h1>
15       </header>
16       <section>
17         {% block contenido %}
18         {% endblock %}
19       </section>
20       <br>
21       <footer>
22         <p><strong>Instituto de Asistencia Social</strong></p>
23         <p>Para más información consultar el
24         <a href="https://www.instasist.org/" target="_blank">Portal del Instituto de Asistencia Social</a>.
25         </p>
26       </footer>
27     </div>
28   </body>
29 </html>
```

[Programación III]

[16]



Django. Primer Proyecto Web

Creación de plantillas o templates – Archivo: base.html

- La etiqueta `{% load static %}` le indica a Django que cargue las etiquetas de plantilla estáticas proporcionadas por la aplicación `django.contrib.staticfiles`, generalmente para manejar archivos css, javascript e imágenes.
- Luego, se puede utilizar la etiqueta `{% static %}` en toda esta plantilla, con la que se pueden incluir los archivos estáticos, tal como la imagen `trabajosocial.jpg`:

```

```

- Se incluyen dos etiquetas `{% block %}` (**titulo y contenido**) para indicar a Django que se desea definir un bloque en esa área. Las plantillas que hereden de este template **base** pueden completar los bloques con contenido.

[Programación III]

[17]



Django. Primer Proyecto Web

Creación de plantillas o templates – Archivo: lista.html

```
1 {% extends "base/base.html" %}
2 {% block titulo %}Lista de Programas{% endblock %}
3 {% block contenido %}
4     <h2>Lista de Programas</h2>
5     <table>
6         <tr>
7             <th>Nombre</th>
8             <th>Tipo Asistencias</th>
9             <th>Fecha Inicio</th>
10            <th>Fecha Fin</th>
11        </tr>
12        {% for programa in programas %}
13            <tr>
14                <td><a href="{% url 'programa:programa_detalle' programa.id %}">{{ programa.nombre }}</a></td>
15                <td><ul>
16                    {% for asistencia in programa.tipo_asistencias.all %}
17                        <li>{{ asistencia }}</li>
18                    {% endfor %}
19                </ul> </td>
20                <td>{{ programa.fecha_inicio|date:"SHORT_DATE_FORMAT" }}</td>
21                <td>{{ programa.fecha_fin|default_if_none:"---" }}</td>
22            </tr>
23        {% endfor %}
24    </table>
25{% endblock %}
```

[Programación III]

[18]

Django. Primer Proyecto Web

Creación de plantillas o templates – Archivo: lista.html

- La etiqueta `{% extends %}` indica a Django que herede de la plantilla `'base/base.html'`
- Los bloques `{% block titulo %}` y `{% block contenido %}` que se definieron en la plantilla **base**, se completan aquí con contenido específico del template.
- Luego se inserta una tabla HTML y su fila de encabezado.
- Las filas se generan dinámicamente, mediante la etiqueta `{% for programa in programas %}` recorriendo el queryset `"programas"`, que se envió como variable de contexto desde la vista al template.
- La celda que corresponde al "nombre del programa" se genera como un enlace a la **vista** detalle de programa, y cuyo destino (url) se define mediante la etiqueta:



[Programación III]

[19]

Django. Primer Proyecto Web

Creación de plantillas o templates – Archivo: detalle.html

```

1  {% extends "base/base.html" %}
2  {% block titulo %}Detalle de Programa{% endblock %}
3  {% block contenido %}
4  <h1>Programa: {{ programa.nombre }}</h1>
5  <p><strong>Tipos de Asistencia</strong></p>
6  <ul>
7      {% for asistencia in programa.tipo_asistencias.all %}
8          <li>{{ asistencia }}</li>
9      {% endfor %}
10 </ul>
11 <p><strong>Fecha de Inicio: </strong>{{ programa.fecha_inicio|date:"SHORT_DATE_FORMAT" }}</p>
12 <p><strong>Fecha de Fin: </strong> {{ programa.fecha_fin|default_if_none: "---" }}</p>
13
14 <form action="{% url 'programa:programa_delete' %}" method="POST">
15     {% csrf_token %}
16     <input type="hidden" value="{{ programa.id }}">
17     <button><a href="{% url 'programa:programa_edit' programa.id %}">Editar</a></button>
18     <input type="submit" value="Eliminar">
19 </form>
20 {% endblock %}
    
```

[Programación III]

[20]

Django. Primer Proyecto Web

Muestra de los templates y vistas creadas

- Ejecutar el proyecto, mediante el comando:
`python manage.py runserver`
para iniciar el servidor de desarrollo.
- Típear la URL
`http://127.0.0.1:8000/programa/`
en el navegador
- Teniendo algunos registros en "programa", se verá algo como esto:



[Programación III]

[21]

Django. Primer Proyecto Web

Muestra de los templates y vistas creadas

- De igual manera, si se tipea la URL
`http://127.0.0.1:8000/programa/1/`
en el navegador, se obtiene la vista de detalle de un programa:

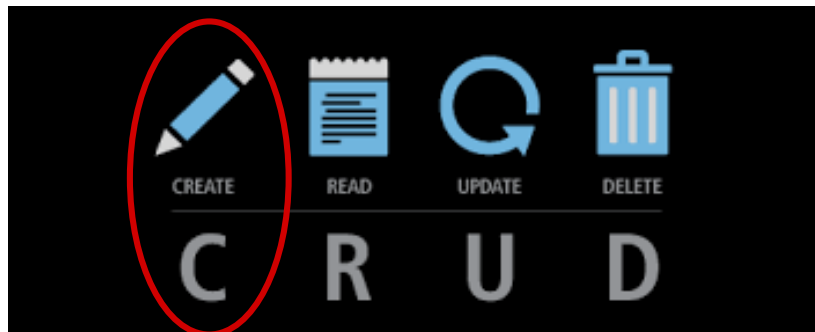


[Programación III]

[22]

Django. Primer Proyecto Web

Operación CREATE



Django. Primer Proyecto Web

Creación de Formularios Django

- Django posee un **framework de formularios** incorporado que permite crear formularios de una manera fácil.
- El framework de formularios simplifica la definición de los campos, especifica cómo deben mostrarse e indica cómo se deben validar los datos de entrada.
- Hay dos clases base para construir formularios:
 - **Form**: permite crear formularios estándares
 - **ModelForm**: permite crear formularios vinculados a instancias de modelos

Referencia: <https://docs.djangoproject.com/en/3.1/ref/forms/>



Django. Primer Proyecto Web

Creación de Formularios para el modelo “Programa”

- Los formularios pueden estar ubicados en cualquier lugar del proyecto Django.
- Pero la convención es colocarlos dentro de un archivo **forms.py** para cada aplicación.
- Para el caso de la creación de un “programa” será necesario utilizar un **ModelForm** porque se tiene que construir un formulario dinámicamente a partir del modelo.
- Crear y editar el archivo **forms.py** de la aplicación de programa:

```
forms.py
8 class ProgramaForm(forms.ModelForm):
9     class Meta:
10         model = Programa
11         fields = ('nombre', 'tipo_asistencias', 'requisitos', 'fecha_inicio', 'fecha_fin')
12
13         widgets = {
14             'requisitos': forms.ClearableFileInput(),
15             'fecha_inicio': DateInput(format='%Y-%m-%d', attrs={'type': 'date'}),
16             'fecha_fin': DateInput(format='%Y-%m-%d', attrs={'type': 'date'})
17         }
```

[Programación III]

[25]



Django. Primer Proyecto Web

Creación de Formularios para el modelo “Programa”

- Para crear un formulario a partir de un modelo, solo se necesita indicar qué modelo usar para construir el formulario en la **clase Meta** del formulario.
- Django realiza una introspección del modelo y crea el formulario de forma dinámica.
- Cada tipo de campo del modelo tiene un tipo de campo de formulario predeterminado correspondiente.
- Se puede indicar explícitamente qué campos se desean incluir en el formulario usando una lista de campos, o definir qué campos desea excluir usando una lista de campos de exclusión.
- También se puede personalizar cómo se muestra un campo de formulario, mediante el atributo **widget**, el cual es la representación de Django de un elemento de entrada HTML.

<https://docs.djangoproject.com/en/3.1/ref/forms/widgets/>

[Programación III]

[26]



Django. Primer Proyecto Web

Manejo de Formularios en las Vistas

- Crear la vista “**programa_create**” en el archivo “**views.py**” de la aplicación **programa**:

```

22 def programa_create(request):
23     nuevo_programa = None
24     if request.method == 'POST':
25         programa_form = ProgramaForm(request.POST, request.FILES)
26         if programa_form.is_valid():
27             # Se guardan los datos que provienen del formulario en la B.D.
28             nuevo_programa = programa_form.save(commit=True)
29             messages.success(request,
30                             'Se ha agregado correctamente el Programa {}'.format(nuevo_programa))
31             return redirect(reverse('programa:programa_detalle', args={nuevo_programa.id}))
32         else:
33             programa_form = ProgramaForm()
34
35     return render(request, 'programa/programa_form.html',
36                 {'form': programa_form})
    
```

[Programación III]

[27]



Django. Primer Proyecto Web

Manejo de Formularios en las Vistas

- Se ha utilizado la misma vista **programa_create** tanto para mostrar el formulario inicial como para procesar los datos enviados.
- Si se recibe una solicitud **GET**, se muestra un formulario vacío, y si recibe una solicitud **POST**, se debe procesar el formulario enviado por el usuario.
- Se usa **request.method == 'POST'** para distinguir entre los dos escenarios.
- Si se trata de una solicitud POST, se crea una instancia de form utilizando los datos enviados que se encuentran en **request.POST** y **request.FILES**.
- Luego, se validan los datos enviados usando el método **is_valid()** del formulario, el cual devuelve el valor True o False según los datos introducidos sean válidos o no.
- Se puede ver una lista de errores de validación accediendo a **form.errors**
- Si el formulario no es válido, se vuelve a representar el formulario en el template con los datos enviados, mostrando los errores de validación.
- Si el formulario es válido, se guardan los datos en la B.D. y se convoca a la vista “programa_detalle” enviando los argumentos correspondientes.

[Programación III]

[28]

Django. Primer Proyecto Web

Representación de formularios en Templates

- Después de crear el formulario, programar la vista y agregar el patrón URL, solo falta la plantilla para esta vista.
- Crear el archivo **“programa_form.html”** en el directorio **programa/templates/programa/** y agregar el siguiente código:

```

1  {% extends "base/base.html" %}
2  {% block titulo %}Creación de Programa{% endblock %}
3  {% block contenido %}
4      <h1>Creación de Programa</h1>
5      <form method="post" enctype="multipart/form-data">
6          {% for field in form %}
7              <div class="campo">
8                  {{ field.errors }}
9                  {{ field.label_tag }} {{ field }}
10             </div><br>
11          {% endfor %}
12          {% csrf_token %}
13          <input type="submit" value="Guardar">
14      </form>
15  {% endblock %}

```

[Programación III]

[29]

Django. Primer Proyecto Web

Representación de formularios en Templates

- Mediante el template anterior se muestra el formulario con los campos de un Programa.
- Solo se debe crear el elemento HTML
<form method="POST" enctype="multipart/form-data">...</form>
- Luego, se debe representar el formulario de Django, lo cual se puede hacer de varias maneras. Una de ellas, es iterando sobre la instancia del formulario y mostrando individualmente las etiquetas HTML **{{ field.label_tag }}**, campos **{{ field }}** y mensajes de error **{{ field.errors }}**
- La etiqueta **{% csrf_token %}** introduce un campo oculto con un token generado automáticamente para evitar ataques de falsificación de solicitudes entre sitios (CSRF). La etiqueta generada queda así:

```

<input type='hidden' name='csrfmiddlewaretoken'
value='26JjKo2lcEtYkGoV9z4XmJIEHLXN5LDR' />

```

[Programación III]

[30]



Django. Primer Proyecto Web

Validación de formularios <https://docs.djangoproject.com/en/3.1/ref/forms/validation/>

- La validación del formulario ocurre cuando se limpian (*clean*) los datos.
- Se ejecutan tres tipos de métodos de limpieza durante el procesamiento del formulario, la mas común es cuando se convoca al método `is_valid()`.
- En general, cualquier método de limpieza puede generar una excepción `ValidationError` si hay un problema con los datos que se están procesando, sino, el método debe devolver los datos limpios (normalizados) como un objeto Python.
- La validación de los datos de un form se realiza en varios pasos, que se pueden personalizar. Lo mas común es sobrescribir los métodos:
 - > `clean()`: se aplica sobre una subclase de **Field** y es responsable de ejecutar otras validaciones y propagar sus errores. Este método devuelve el diccionario `clean_data` del formulario si todo se ha validado correctamente.
 - > `clean_<fieldname>`: realiza una limpieza específica de un atributo en particular. El valor del campo se encuentra en el diccionario `self.cleaned_data` ya que previamente se ha ejecutado el método `clean()`

[Programación III]

[31]



Django. Primer Proyecto Web

Validación de formularios

- A continuación, realizaremos dos validaciones en el formulario de programa:
 - 1) que la fecha de inicio del programa no sea posterior a la fecha fin.
 - 2) que el tipo de archivo que se cargue en requisitos sea de tipo "pdf"

```
forms.py
16 def clean(self):
17     cleaned_data = super().clean()
18     fecha_inicio = self.cleaned_data['fecha_inicio']
19     fecha_fin = self.cleaned_data['fecha_fin']
20     # Verifica que la fecha de inicio sea anterior a fecha fin.
21     if fecha_fin and fecha_inicio > fecha_fin:
22         raise ValidationError(
23             {'fecha_inicio': 'La Fecha de Inicio no puede ser posterior que la fecha fin'},
24             code='invalido'
25         )
26     return cleaned_data
27
28 def clean_requisitos(self):
29     requisitos = self.cleaned_data['requisitos']
30     if requisitos:
31         extension = requisitos.name.split('.', 1)[1].lower()
32         if extension != 'pdf':
33             raise forms.ValidationError('El archivo seleccionado no tiene el formato PDF.')
34     return requisitos
```

[Programación III]

[32]



Django. Primer Proyecto Web

Validación de formularios

- Para validar las **fechas** se debe sobrescribir el método **clean()**, ya que es necesario acceder a varios campos del formulario.
- La variable **cleaned_data** es un diccionario que contendrá todos los campos del formulario con una validación previa (ej: que el valor del campo coincida con el tipo de dato definido), por esto, se convoca al método **super().clean()** para asegurarnos que se mantenga cualquier lógica de validación en las clases principales.
- Luego, se obtiene el valor del campo sanitizado accediendo a un elemento del diccionario **cleaned_data** ingresando como clave, el nombre del campo correspondiente.
- Cuando se hace la comparación de fechas de acuerdo a la restricción dada, si ésta no se cumple, se debe devolver una excepción **ValidationError**, enviando como parámetro un diccionario con el nombre del campo y descripción del error, y otro argumento **code='invalido'**.
- En el caso de que se cumpla la validación, se debe devolver el diccionario **cleaned_data** con los campos sanitizados.

[Programación III]

[33]



Django. Primer Proyecto Web

Validación de formularios

- Para validar el campo "**requisitos**" se puede sobrescribir el método **clean_requisitos()**, ya que solo debemos acceder a ese campo.
- En este caso, se accede al valor sanitizado del campo mediante la variable **self.cleaned_data['requisitos']**.
- Luego, si existe un valor para requisitos (si se cargó un archivo), se accede a la propiedad **name** que contiene el nombre del archivo junto con su extensión.
- Mediante la función **rsplit** podemos obtener la parte de la extensión del archivo y luego preguntar si el valor es "pdf", caso contrario se lanza una excepción.

[Programación III]

[34]

Django. Primer Proyecto Web

Validación de formularios

- Las validaciones de los campos se muestran de esta manera:

Sistema de Gestión de Asistencias Sociales

Creación de Programa

Nombre:

Tipo asistencias:

- El archivo seleccionado no tiene el formato PDF.

Requisitos: links.txt

- La Fecha de Inicio no puede ser posterior que la fecha fin

Fecha inicio:

Fecha fin:

[Programación III]

[35]

Django. Primer Proyecto Web

Validación de formularios

- Si se guardan los datos de un programa, la vista “programa_create” nos dirige a la vista “programa_detalle”, enviando un mensaje de retroalimentación:

Sistema de Gestión de Asistencias Sociales

Programa: Asistencia Social

Se ha agregado correctamente el Programa Asistencia Social

Tipos de Asistencia

- Ayuda monetaria

Fecha de Inicio: 01/10/2020

Fecha de Fin: ---

Requisitos: No posee

Instituto de Asistencia Social

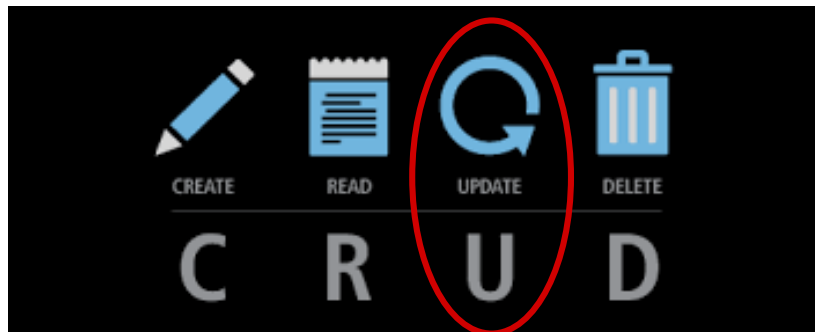
Para más información consultar el [Portal del Instituto de Asistencia Social](#).

[Programación III]

[36]

Django. Primer Proyecto Web

Operación UPDATE



[Programación III]

[37]

Django. Primer Proyecto Web

Creación de Vista para actualizar un programa

- En el archivo “view.py” de la app “programa” agregar la siguiente vista:

```
43 def programa_edit(request, pk):
44     programa = get_object_or_404(Programa, pk=pk)
45     if request.method == 'POST':
46         form_programa = ProgramaForm(request.POST, request.FILES, instance=programa)
47         if form_programa.is_valid():
48             form_programa.save()
49             messages.success(request, 'Se ha actualizado correctamente el Programa')
50             return redirect(reverse('programa:programa_detalle', args=[programa.id]))
51     else:
52         form_programa = ProgramaForm(instance=programa)
53
54     return render(request, 'programa/programa_edit.html', {'form': form_programa})
```

[Programación III]

[38]



Django. Primer Proyecto Web

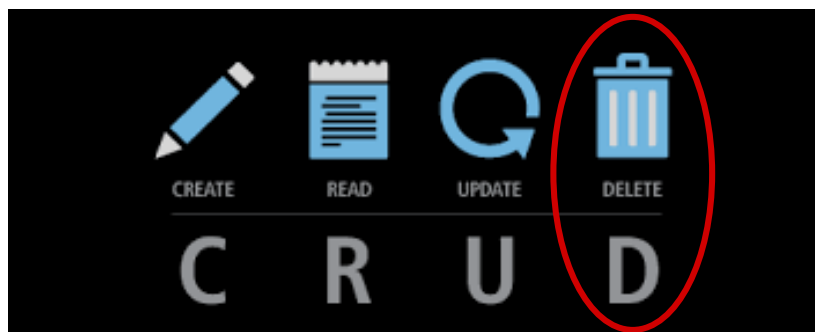
Creación de Vista para actualizar un programa

- En la vista anterior, se recupera un programa mediante su ID.
- Si el método de la petición de la vista es igual **GET**, se crea una instancia de la clase **ProgramaForm**, enviando como parámetro, la instancia del programa recuperado. Esto le indica a Django que se deben cargar los campos del formulario, con los datos de un programa en particular.
- Si la petición es igual a **POST**, se crea una instancia del formulario, con los datos que provienen del formulario html, junto con el argumento **"instance"**, lo cual permite que el método **save()** del formulario, ejecute una sentencia SQL UPDATE sobre el programa recuperado.



Django. Primer Proyecto Web

Operación DELETE





Django. Primer Proyecto Web

Creación de Vista para eliminar un programa

- En el archivo “view.py” de la app “programa” agregar la siguiente vista:

```
views.py
39 def programa_delete(request):
40     if request.method == 'POST':
41         if 'id_programa' in request.POST:
42             programa = get_object_or_404(Programa, pk=request.POST['id_programa'])
43             nombre_programa = programa.nombre
44             programa.delete()
45             messages.success(request, 'Se ha eliminado exitosamente el Programa {}'.format(nombre_programa))
46         else:
47             messages.error(request, 'Debe indicar qué Programa se desea eliminar')
48     return redirect(reverse('programa:programa_lista'))
```



Django. Primer Proyecto Web

Creación de Vista para eliminar un programa

- En la vista anterior, se guarda en una variable una instancia de un programa mediante su ID, el cual fue enviado a través del método POST desde un formulario web.
- Luego se recupera el nombre del programa, para luego mostrarlo en un mensaje que se enviará a la vista después de haber eliminado la instancia de programa.
- Finalmente, la vista redirige el flujo de control a la vista **programa_lista** donde se muestran todos los programas existentes.

Django. Primer Proyecto Web

Creación de una página principal (HOME)

- En toda Webapp existe una pagina principal asociada a la URL raiz.
- Para lograr esto, crear un documento HTML al que llamaremos "home.html" dentro del directorio "templates" del proyecto.
- Esta página Home puede o no heredar de la plantilla base.
- Para asociar la url de la petición con la pagina home.html, agregar un patrón de URL dentro del archivo urls.py del proyecto:

```
urlpatterns = [
    ...
    path("", TemplateView.as_view(template_name='base/home.html'), name='home'),
]
```

Como no se necesita mayor procesamiento de datos, se puede utilizar esta clase de vista, para renderizar un template dado

[Programación III]

[43]

Django. Primer Proyecto Web

Creación de una página principal (HOME)

- Finalmente, tipear en el navegador la URL: <http://127.0.0.1:8000/>



[Programación III]

[44]



Grails. Actividad 4.1 – Caso estudio: Asistencias



- Dado el caso de estudio de la aplicación web “Asistencias”, continuar con la definición de modelos, vistas y templates correspondientes de las aplicaciones “programa” y “persona”
- Consultar el proyecto en el repositorio git, rama Master:
<https://github.com/unca-programacion3/asistencias.git>



GRACIAS POR TU ATENCIÓN!!

DOCENTES DE CATEDRA:

- ❖ Mgtr. Cecilia E. Gallardo
- ❖ Esp. Marta Miranda

