Grado en Ingeniero en Informática
Programación Web
Febrero 2016

Examen de preguntas de respuesta corta

Responda de forma breve y razonada a las siguientes cuestiones:

1. Explique qué hace y cómo funciona el siguiente segmento de código de Django:

```
def funcion(request):
    if 'query' in request.GET:
        distilleries =  Distillery.objects.filter(name__icontains = request.GET['query'])
        letters = string.ascii_uppercase
        context = {'distilleries': distilleries, 'letters': letters}

        return render(request, "bottles/distilleries.html", context)
    else:
        message = 'You submitted an empty form.'
        context = {'message': message}
        return render(request, "bottles/error_message.html", context)
```

   RESPUESTA: Se trata de la vista que acompaña a un formulario de búsqueda. Se buscan aquellas botellas que contienen el texto introducido en el formulario.

2. Para proteger una vista en Django para evitar que un usuario no autenticado acceda a ella, para el caso de vistas basadas en clases, existen dos alternativas. Podemos decorar el resultado del método `as_view()`:

```
urlpatterns = patterns('',
    (r'^about/', login_required(TemplateView.as_view(template_name="secret.html"))),
    (r'^vote/', permission_required('polls.can_vote')(VoteView.as_view())),
)
```

   o podemos decorar el método `dispatch()` de la clase:

```
class ProtectedView(TemplateView):
    template_name = 'secret.html'
    @method_decorator(login_required)
    def dispatch(self, *args, **kwargs):
        return super(ProtectedView, self).dispatch(*args, **kwargs)
```

   Indique qué diferencias hay entre estas dos opciones.

   RESPUESTA: En el primer caso solo protegemos el acceso si se hace con la dirección URL concreta, pero no si se hace desde otra que haga referencia a la misma vista. En el segundo caso se protege el acceso en cualquier caso.

3. ¿Qué ventajas tienen los nuevos elementos semánticos introducidos en HTML5 respecto a los tradicionales de HTML4?

   RESPUESTA: Permite una mejor organización de los contenidos de una página, ya que estos se pueden introducir en bloques semánticos.

4. ¿Qué diferencia existe entre los dos ejemplos siguientes de captura de URLs?

```
urlpatterns = patterns('',
    url(r'^articles/(\d{4})/$', views.year_archive),
)

urlpatterns = patterns('',
    url(r'^articles/(?P<year>\d{4})/$', views.year_archive),
)
```

RESPUESTA: Ambas expresiones filtrarían la misma URL, pero en el primero caso se pasa el argumento numérico como argumento posicional, en el segundo caso se le nombra como `year`.

5. Aunque desde una plantilla se pueden llamar a los métodos de un objeto, ¿qué limitación pone Django a los métodos que se pueden llamar desde la plantilla?

   REPUESTA: Los métodos llamados desde una plantilla no pueden recibir ningún argumento adicional al objeto de la clase correspondiente.

6. Considere la siguiente clase en Django que modela un equipo de fútbol y considere que la clase Jugador está ya definida:

```
class Equipo(models.Model):
    nombre = models.CharField(max_length = 100)
    capitan = models.ForeignKey(Jugador)
    plantilla = models.ManyToManyField(Jugador)
```

   Cuando intentamos generar el modelo correspondiente en la base de datos obtendríamos un error, ¿cuál es el error y cómo lo podríamos solucionar?

   RESPUESTA: La clase `Equipo` tiene dos relaciones, una 1:N y otra N:N, con la clase `Jugador`. Esto es perfectamente válido pero genera un problema en Django. La relación inversa se crea por defecto cada vez que creamos una relación así, de esta forma, una campo con el nombre `equipo` sería creado dos veces en la clase `Jugador`. Podemos evitar el problema añadiendo la opción:

```
class Equipo(models.Model):
    nombre = models.CharField(max_length = 100)
    capitan = models.ForeignKey(Jugador, related_name="capitanes")
    plantilla = models.ManyToManyField(Jugador, related_name="jugadores")
```

   O podemos elegir no crear la relación inversa con la opción: `related_name='+'`.

7. Suponga que tenemos una aplicación, `app1`, en Django que contiene el siguiente patrón URL:

```
from django.conf.urls import patterns, url
from app1.views import archive
urlpatterns = patterns('',
    url(r'^archive-summary/(\d{4})/$', archive, name="list"),
)
```

   Y una segunda aplicación, `app2`, dentro del mismo proyecto:

```
from django.conf.urls import patterns, url
from app2.views import archive
urlpatterns = patterns('',
    url(r'^news-list/(\d{4})/$', archive, name="list"),
)
```

¿Es posible distinguir en una plantilla HTML las dos URLs a pesar de que su nombre es el mismo?

RESPUESTA: Sí, podemos usar un espacio de nombres:

```
url(r'^app1/', include('app1.urls', namespace = "app1"))

href="{% url 'app1:list' year %}"
```

8. ¿Qué diferencias de funcionamiento existirían entre las dos implementaciones de la función `detail()` siguientes en Django?

```
def detail(request, question_id):
    question = Question.objects.get(pk=question_id)
    return render(request, 'polls/detail.html', {'question': question})

def detail(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    return render(request, 'polls/detail.html', {'question': question})
```

RESPUESTA: En el primer caso si el objeto accedido no existe en la base de datos habrá un error que no está considerando y el servicio web se suspenderá. En el segundo caso el error será capturado y se generará una excepción 404.

9. ¿Qué diferencia hay entre estos dos segmentos de código?

```
a = Author()
f = AuthorForm(request.POST, instance=a)
```

y

```
f = AuthorForm(request.POST)
```

RESPUESTA: En el primer caso se le pasa al formulario una instancia del objeto por lo que se rellenaría inicialmente con esos valores y tendríamos una vista de edición. En el segundo caso esto no se hace y tenemos una vista para crear un objeto nuevo.

10. ¿Qué haría la siguiente vista?

```
# views.py
from django.views.generic import ListView
from books.models import Publisher

class PublisherList(ListView):
    model = Publisher
```

RESPUESTA: Es una vista basada en clases que nos proporcionaría la lista de objetos de la clase `Publisher` y se la pasaría a la correspondiente plantilla en el objeto `object_list`.

Tiempo de realización: **1** hora. Todas las cuestiones tienen la misma calificación de 1 punto.