**Opciones de campo**

Los siguientes argumentos están disponibles para todos los tipos de campo.

Todos son opcionales.

Valor nulo(booleano):

Puede tomar solo una opción(true o false), y nos indica que el campo puede ser nulo, ósea no declarar ningún valor para que nos tire un error

Declaración:

nombre= models.CharField(*max\_length*=12, *null*=True)

valor blanco(booleano):

Es similar al valor nulo, solo que el valor nulo se relaciona con la base de datos, en cambio en valor blanco se relaciona con la validacion

Si un campo tiene en blanco = Verdadero, la validación del formulario permitirá la entrada de un valor vacío. Si un campo tiene en blanco = Falso, el campo será obligatorio:

nombre = models.CharField(*max\_length*=255, *blank*=True)

choices(tupla):

nos permite elegir multiples opciones para nuestro campo definido, por ejemplo:

*class* book(*models*.*Model*):

    Book\_choices = [

        ('las cronicas de narnia', 'las cronicas de narnia'),

        ('harry potter', 'harry potter'),

        ('rapido y furioso', 'rapido y furioso')

    ]

    title = models.CharField(*max\_length*=30, *choices*=Book\_choices)

db\_column:

El nombre de la columna de la base de datos que se utilizará para este campo. Si no se proporciona, Django usará el nombre del campo.

nombre = models.CharField(*max\_length*=30, *db\_column*='name')

este atributo si visualizara como ‘name’ en nuestra base de datos

db\_index(booleano)**:**

Si es True, se creará un índice de base de datos para este campo.

 nombre = models.CharField(*max\_length*=30, *db\_index*=True)

db\_tablespace(booleano):

permite incorporar espacios en blanco en nuestro campo, si esta opcion no esta habilitado, ignorada los espacios en blanco

 nombre = models.CharField(*max\_length*=30, *db\_tablespace*=True)

default:

agrega por defecto un valor aleatorio al campo, es para tener una mejor noción del contenido que hay que agregar al mismo

nombre = models.CharField(*max\_length*=30, *default*='leandro')

editable(booleano):

Si es False, el campo no se mostrará en el administrador ni en ningún otro ModelForm. También se omiten durante la validación del modelo.

El valor predeterminado es True.

help\_text:

Texto de «ayuda» adicional que se mostrará con el widget de formulario. Es útil para la documentación incluso si su campo no se usa en un formulario.

fecha = models.DateTimeField(*help\_text*='por favor, usar este forma como: YYYY-MM-DD')

primary\_key(booleano):

Si es verdadero, este campo es la clave principal del modelo.

Solo se permite una clave principal en un objeto.

El campo de clave principal es de solo lectura. Si cambia el valor de la clave principal en un objeto existente y luego lo guarda, se creará un nuevo objeto junto con el anterior.

  matricula=models.integerField(*primary\_key*=True)

unique(booleano):

Si es True, este campo debe ser único en toda la tabla

Esto se aplica a nivel de la base de datos y mediante la validación del modelo. Si intenta guardar un modelo con un valor duplicado en un campo único, el método save () del modelo generará un django.db.IntegrityError.

Esta opción es válida en todos los tipos de campo **excepto** ManyToManyField y OneToOneField.

Tenga en cuenta que cuando único es Verdadero, no es necesario especificar db\_index, porque único implica la creación de un índice.

matricula=models.integerField(*unique*=True)

unique\_for\_date(booleano):

Establezca esto en el nombre de DateField o DateTimeField para requerir que este campo sea único para el valor del campo de fecha.

Por ejemplo, si tiene un título de campo que tiene unique\_for\_date = "pub\_date", Django no permitirá la entrada de dos registros con el mismo título y pub\_date.

matricula=models.DateTimeField(*unique\_for\_date*=True)

ademas, podemos especificar por mes y año con los comandos:

unique\_for\_month

unique\_for\_year

verbose\_name(‘nombre’):

Un nombre legible por humanos para el campo. Si no se proporciona el nombre detallado, Django lo creará automáticamente usando el nombre del atributo del campo, convirtiendo los guiones bajos en espacios. Consulte Nombres de campo detallados.

Validators:

Una lista de validadores que se ejecutarán para este campo

Antes que nada necesitamos importar las librerías para acceder a estas validaciones, de la forma:

from django.core.validators import

El módulo **django.core.validators** contiene una colección de validadores invocables para usar con campos de formulario y modelo. Se usan internamente, pero también están disponibles para usar con sus propios campos.

Se pueden usar además o en lugar de los métodos field.clean () personalizadosen la parte de ‘import’ podemos agregar las siguientes validaciones:

### validate\_email: