**PATH : ProjectName\AppName\models.py**

**Define Field types**

**class Person(models.Model):**

**per\_name = models.CharField(max\_length=24)**

**father\_name =models.CharField(max\_length=200)**

**gender =models.CharField(max\_length=22,choices=GENDER\_CHOICES,default='Male')**

**mobile = models.CharField(max\_length=10)**

**per\_address =models.TextField(max\_length=555,default=None)**

**per\_pincode = models.CharField(max\_length=6,default=None)**

**per\_photo = models.FileField(upload\_to='images/',default=None)**

**per\_status = models.BooleanField('Are You Married',default=True)**

**publication\_date = models.DateField()**

def \_\_str\_\_(self):

return self.per\_name

**Define Field types with Hint’s**

**city\_name = models.CharField(max\_length=30,help\_text='Enter City Name')**

**publication\_date = models.DateField(help\_text='yy/mm/dd')**

**Passing parameter within Fieldstype as argument.**

**Ex:**

* [help\_text](https://docs.djangoproject.com/en/2.1/ref/models/fields/#help-text): Provides a text label for HTML forms (e.g. in the admin site), as described above.
* [verbose\_name](https://docs.djangoproject.com/en/2.1/ref/models/fields/#verbose-name): A human-readable name for the field used in field labels. If not specified, Django will infer the default verbose name from the field name.
* [default](https://docs.djangoproject.com/en/2.1/ref/models/fields/#default): The default value for the field. This can be a value or a callable object, in which case the object will be called every time a new record is created.
* [null](https://docs.djangoproject.com/en/2.1/ref/models/fields/#null): If True, Django will store blank values as NULL in the database for fields where this is appropriate (a CharField will instead store an empty string). The default is False.
* [blank](https://docs.djangoproject.com/en/2.1/ref/models/fields/#blank): If True, the field is allowed to be blank in your forms. The default is False, which means that Django's form validation will force you to enter a value. This is often used with null=True , because if you're going to allow blank values, you also want the database to be able to represent them appropriately.
* [choices](https://docs.djangoproject.com/en/2.1/ref/models/fields/#choices): A group of choices for this field. If this is provided, the default corresponding form widget will be a select box with these choices instead of the standard text field.
* [primary\_key](https://docs.djangoproject.com/en/2.1/ref/models/fields/#primary-key): If True, sets the current field as the primary key for the model (A primary key is a special database column designated to uniquely identify all the different table records). If no field is specified as the primary key then Django will automatically add a field for this purpose.

There are many other options — you can view the [full list of field options here](https://docs.djangoproject.com/en/2.1/ref/models/fields/#field-options).

<https://docs.djangoproject.com/en/2.1/ref/models/fields/#field-options>

##### Common field types

The following list describes some of the more commonly used types of fields.

* [CharField](https://docs.djangoproject.com/en/2.1/ref/models/fields/#django.db.models.CharField) is used to define short-to-mid sized fixed-length strings. You must specify the max\_length of the data to be stored.
* [TextField](https://docs.djangoproject.com/en/2.1/ref/models/fields/#django.db.models.TextField) is used for large arbitrary-length strings. You may specify a max\_length for the field, but this is used only when the field is displayed in forms (it is not enforced at the database level).
* [IntegerField](https://docs.djangoproject.com/en/2.1/ref/models/fields/#django.db.models.IntegerField) is a field for storing integer (whole number) values, and for validating entered values as integers in forms.
* [DateField](https://docs.djangoproject.com/en/2.1/ref/models/fields/#datefield) and [DateTimeField](https://docs.djangoproject.com/en/2.1/ref/models/fields/" \l "datetimefield) are used for storing/representing dates and date/time information (as Python datetime.date in and datetime.datetime objects, respectively). These fields can additionally declare the (mutually exclusive) parameters auto\_now=True (to set the field to the current date every time the model is saved), auto\_now\_add (to only set the date when the model is first created) , and default (to set a default date that can be overridden by the user).
* [EmailField](https://docs.djangoproject.com/en/2.1/ref/models/fields/#emailfield) is used to store and validate email addresses.
* [FileField](https://docs.djangoproject.com/en/2.1/ref/models/fields/#filefield) and [ImageField](https://docs.djangoproject.com/en/2.1/ref/models/fields/" \l "imagefield) are used to upload files and images respectively (the ImageField simply adds additional validation that the uploaded file is an image). These have parameters to define how and where the uploaded files are stored.
* [AutoField](https://docs.djangoproject.com/en/2.1/ref/models/fields/#autofield) is a special type of IntegerField that automatically increments. A primary key of this type is automatically added to your model if you don’t explicitly specify one.
* [ForeignKey](https://docs.djangoproject.com/en/2.1/ref/models/fields/#foreignkey) is used to specify a one-to-many relationship to another database model (e.g. a car has one manufacturer, but a manufacturer can make many cars). The "one" side of the relationship is the model that contains the "key" (models containing a "foreign key" referring to that "key", are on the "many" side of such a relationship).
* [ManyToManyField](https://docs.djangoproject.com/en/2.1/ref/models/fields/#manytomanyfield) is used to specify a many-to-many relationship (e.g. a book can have several genres, and each genre can contain several books). In our library app we will use these very similarly to ForeignKeys, but they can be used in more complicated ways to describe the relationships between groups. These have the parameter on\_delete to define what happens when the associated record is deleted (e.g. a value of models.SET\_NULL would simply set the value to NULL).

**Model Lavel Meta Class Options**

class City(models.Model):

city\_name = models.CharField(max\_length=30,help\_text='Enter City Name')

city\_code = models.CharField(max\_length=4)

class Meta:

ordering = ['city\_name'] # Works same As Order Clouse like mysql

verbose\_name = ‘ ’ # I don’t know

def \_\_str\_\_(self):

return self.city\_name

The full list of metadata options are available here: [Model metadata options](https://docs.djangoproject.com/en/2.1/ref/models/options/) (Django docs).

<https://docs.djangoproject.com/en/2.1/ref/models/fields/#field-options>

#### Methods

A model can also have methods.

**Minimally, in every model you should define the standard Python class method \_\_str\_\_() to return a human-readable string for each object.** This string is used to represent individual records in the administration site (and anywhere else you need to refer to a model instance). Often this will return a title or name field from the model.

def \_\_str\_\_(self):

  return self.field\_name