**Creating objects**

To create an object, instantiate it using keyword arguments to the model class, then call [**save()**](https://docs.djangoproject.com/en/3.0/ref/models/instances/#django.db.models.Model.save) to save it to the database.

Assuming models live in a file **mysite/blog/models.py**, here’s an example:

**>>> from** **blog.models** **import** Blog

**>>>** b = Blog(name='Beatles Blog', tagline='All the latest Beatles news.')

**>>>** b.save()

This performs an **INSERT** SQL statement behind the scenes. Django doesn’t hit the database until you explicitly call [**save()**](https://docs.djangoproject.com/en/3.0/ref/models/instances/#django.db.models.Model.save).

The [**save()**](https://docs.djangoproject.com/en/3.0/ref/models/instances/#django.db.models.Model.save) method has no return value.

**Saving changes to objects**

To save changes to an object that’s already in the database, use [**save()**](https://docs.djangoproject.com/en/3.0/ref/models/instances/#django.db.models.Model.save).

Given a **Blog** instance **b5** that has already been saved to the database, this example changes its name and updates its record in the database:

**>>>** b5.name = 'New name'

**>>>** b5.save()

This performs an **UPDATE** SQL statement behind the scenes. Django doesn’t hit the database until you explicitly call [**save()**](https://docs.djangoproject.com/en/3.0/ref/models/instances/#django.db.models.Model.save).

**Saving ForeignKey and ManyToManyField fields**

Updating a [**ForeignKey**](https://docs.djangoproject.com/en/3.0/ref/models/fields/#django.db.models.ForeignKey) field works exactly the same way as saving a normal field – assign an object of the right type to the field in question. This example updates the **blog** attribute of an **Entry** instance **entry**, assuming appropriate instances of **Entry** and **Blog** are already saved to the database (so we can retrieve them below):

**>>> from** **blog.models** **import** Blog, Entry

**>>>** entry = Entry.objects.get(pk=1)

**>>>** cheese\_blog = Blog.objects.get(name="Cheddar Talk")

**>>>** entry.blog = cheese\_blog

**>>>** entry.save()

Updating a [**ManyToManyField**](https://docs.djangoproject.com/en/3.0/ref/models/fields/#django.db.models.ManyToManyField) works a little differently – use the [**add()**](https://docs.djangoproject.com/en/3.0/ref/models/relations/#django.db.models.fields.related.RelatedManager.add) method on the field to add a record to the relation. This example adds the **Author** instance **joe** to the **entry** object:

**>>> from** **blog.models** **import** Author

**>>>** joe = Author.objects.create(name="Joe")

**>>>** entry.authors.add(joe)

To add multiple records to a [**ManyToManyField**](https://docs.djangoproject.com/en/3.0/ref/models/fields/#django.db.models.ManyToManyField) in one go, include multiple arguments in the call to [**add()**](https://docs.djangoproject.com/en/3.0/ref/models/relations/#django.db.models.fields.related.RelatedManager.add), like this:

**>>>** john = Author.objects.create(name="John")

**>>>** paul = Author.objects.create(name="Paul")

**>>>** george = Author.objects.create(name="George")

**>>>** ringo = Author.objects.create(name="Ringo")

**>>>** entry.authors.add(john, paul, george, ringo)

Django will complain if you try to assign or add an object of the wrong type.