2.4. Documents instances

To create a new document object, create an instance of the relevant document class, providing values for its fields as constructor keyword arguments. You may provide values for any of the fields on the document:

```
>>> page = Page(title="Test Page")
>>> page.title
'Test Page'
```

You may also assign values to the document's fields using standard object attribute syntax:

```
>>> page.title = "Example Page"
>>> page.title
'Example Page'
```

2.4.1. Saving and deleting documents

MongoEngine tracks changes to documents to provide efficient saving. To save the document to the database, call the `save()` method. If the document does not exist in the database, it will be created. If it does already exist, then any changes will be updated atomically. For example:

```
>>> page = Page(title="Test Page")
>>> page.save()  # Performs an insert
>>> page.title = "My Page"
>>> page.save()  # Performs an atomic set on the title field.
```

Note

Changes to documents are tracked and on the whole perform `set` operations.

`list_field.push(0)` — *sets* the resulting list
`del(list_field)` — *unsets* whole list
With lists its preferable to use `Doc.update(push__list_field=0)` as this stops the whole list being updated — stopping any race conditions.

See also

Atomic updates

2.4.1.1. Cascading Saves

If your document contains `ReferenceField` or `GenericReferenceField` objects, then by default the `save()` method will not save any changes to those objects. If you want all references to be saved also, noting each save is a separate query, then passing `cascade` as True to the save method will cascade any saves.

2.4.1.2. Deleting documents

To delete a document, call the `delete()` method. Note that this will only work if the document exists in the database and has a valid `id`.

2.4.2. Document IDs

Each document in the database has a unique id. This may be accessed through the `id` attribute on `Document` objects. Usually, the id will be generated automatically by the database server when the object is save, meaning that you may only access the `id` field once a document has been saved:

```
>>> page = Page(title="Test Page")
>>> page.id
>>> page.save()
>>> page.id
ObjectId('123456789abcdef000000000')
```

Alternatively, you may define one of your own fields to be the document's "primary key" by providing `primary_key=True` as a keyword argument to a field's constructor. Under the hood, MongoEngine will use this field as the `id`; in fact `id` is actually aliased to your primary key field so you may still use `id` to access the primary key if you want:

```
>>> class User(Document):
...     email = StringField(primary_key=True)
...     name = StringField()
...
>>> bob = User(email='bob@example.com', name='Bob')
>>> bob.save()
>>> bob.id == bob.email == 'bob@example.com'
True
```

You can also access the document's "primary key" using the `pk` field, it's an alias to `id`:

```
>>> page = Page(title="Another Test Page")
>>> page.save()
>>> page.id == page.pk
True
```

Note

If you define your own primary key field, the field implicitly becomes required, so a `ValidationError` will be thrown if you don’t provide it.