**marshmallow** is an ORM/ODM/framework-agnostic library for converting complex datatypes, such as objects, to and from native Python datatypes.

In short, marshmallow schemas can be used to:

* **Validate** input data.
* **Deserialize** input data to app-level objects.
* **Serialize** app-level objects to primitive Python types. The serialized objects can then be rendered to standard formats such as JSON for use in an HTTP API.

Declaring Schemas

Let’s start with a basic user “model”.

**import** datetime **as** dt

**class** User**:**

**def** \_\_init\_\_**(**self**,** name**,** email**):**

self.name = name

self.email = email

self.created\_at = dt.datetime.now**()**

**def** \_\_repr\_\_**(**self**):**

**return** "<User(name={self.name!r})>".format**(**self=self**)**

Create a schema by defining a class with variables mapping attribute names to [**Field**](https://marshmallow.readthedocs.io/en/stable/marshmallow.fields.html#marshmallow.fields.Field) objects.

**from** marshmallow **import** Schema**,** fields

**class** UserSchema**(**Schema**):**

name = fields.Str**()**

email = fields.Email**()**

created\_at = fields.DateTime**()**

## Serializing Objects (“Dumping”)

Serialize objects by passing them to your schema’s [**dump**](https://marshmallow.readthedocs.io/en/stable/api_reference.html#marshmallow.Schema.dump) method, which returns the formatted result.

**from** pprint **import** pprint

user = User**(**name="Monty"**,** email="monty@python.org"**)**

schema = UserSchema**()**

result = schema.dump**(**user**)**

pprint**(**result**)**

*# {"name": "Monty",*

*# "email": "monty@python.org",*

*# "created\_at": "2014-08-17T14:54:16.049594+00:00"}*

You can also serialize to a JSON-encoded string using [**dumps**](https://marshmallow.readthedocs.io/en/stable/api_reference.html#marshmallow.Schema.dumps).

json\_result = schema.dumps**(**user**)**

pprint**(**json\_result**)**

*# '{"name": "Monty", "email": "monty@python.org", "created\_at": "2014-08-17T14:54:16.0*

## Filtering Output

You may not need to output all declared fields every time you use a schema. You can specify which fields to output with the only parameter.

summary\_schema = UserSchema**(**only=**(**"name"**,** "email"**))**

summary\_schema.dump**(**user**)**

*# {"name": "Monty", "email": "monty@python.org"}*

You can also exclude fields by passing in the exclude parameter.

## Deserializing Objects (“Loading”)

The reverse of the [**dump**](https://marshmallow.readthedocs.io/en/stable/api_reference.html#marshmallow.Schema.dump) method is [**load**](https://marshmallow.readthedocs.io/en/stable/api_reference.html#marshmallow.Schema.load), which validates and deserializes an input dictionary to an application-level data structure.

By default, [**load**](https://marshmallow.readthedocs.io/en/stable/api_reference.html#marshmallow.Schema.load) will return a dictionary of field names mapped to deserialized values (or raise a **[ValidationError](https://marshmallow.readthedocs.io/en/stable/marshmallow.exceptions.html" \l "marshmallow.exceptions.ValidationError" \o "marshmallow.exceptions.ValidationError)** with a dictionary of validation errors, which we’ll [revisit later](https://marshmallow.readthedocs.io/en/stable/quickstart.html#validation)).

**from** pprint **import** pprint

user\_data = **{**

"created\_at"**:** "2014-08-11T05:26:03.869245"**,**

"email"**:** "ken@yahoo.com"**,**

"name"**:** "Ken"**,**

**}**

schema = UserSchema**()**

result = schema.load**(**user\_data**)**

pprint**(**result**)**

### **Deserializing to Objects**[**¶**](https://marshmallow.readthedocs.io/en/stable/quickstart.html#deserializing-to-objects)

In order to deserialize to an object, define a method of your [**Schema**](https://marshmallow.readthedocs.io/en/stable/api_reference.html#marshmallow.Schema) and decorate it with **[post\_load](https://marshmallow.readthedocs.io/en/stable/marshmallow.decorators.html" \l "marshmallow.decorators.post_load" \o "marshmallow.decorators.post_load)**. The method receives a dictionary of deserialized data.

**from** marshmallow **import** Schema**,** fields**,** post\_load

**class** UserSchema**(**Schema**):**

name = fields.Str**()**

email = fields.Email**()**

created\_at = fields.DateTime**()**

@post\_load

**def** make\_user**(**self**,** data**,** \*\*kwargs**):**

**return** User**(**\*\*data**)**

Now, the [**load**](https://marshmallow.readthedocs.io/en/stable/api_reference.html#marshmallow.Schema.load) method return a User instance.

user\_data = **{**"name"**:** "Ronnie"**,** "email"**:** "ronnie@stones.com"**}**

schema = UserSchema**()**

result = schema.load**(**user\_data**)**

print**(**result**)** *# => <User(name='Ronnie')>*

## Validation

[**Schema.load()**](https://marshmallow.readthedocs.io/en/stable/api_reference.html#marshmallow.Schema.load) (and its JSON-decoding counterpart, **[Schema.loads()](https://marshmallow.readthedocs.io/en/stable/api_reference.html" \l "marshmallow.Schema.loads" \o "marshmallow.Schema.loads)**) raises a **[ValidationError](https://marshmallow.readthedocs.io/en/stable/marshmallow.exceptions.html" \l "marshmallow.exceptions.ValidationError" \o "marshmallow.exceptions.ValidationError)** error when invalid data are passed in. You can access the dictionary of validation errors from the **ValidationError.messages** attribute. The data that were correctly deserialized are accessible in **ValidationError.valid\_data**. Some fields, such as the [**Email**](https://marshmallow.readthedocs.io/en/stable/marshmallow.fields.html#marshmallow.fields.Email) and [**URL**](https://marshmallow.readthedocs.io/en/stable/marshmallow.fields.html#marshmallow.fields.URL) fields, have built-in validation.

**from** pprint **import** pprint

**from** marshmallow **import** Schema**,** fields**,** ValidationError

**class** BandMemberSchema**(**Schema**):**

name = fields.String**(**required=**True)**

email = fields.Email**()**

user\_data = **[**

**{**"email"**:** "mick@stones.com"**,** "name"**:** "Mick"**},**

**{**"email"**:** "invalid"**,** "name"**:** "Invalid"**},** *# invalid email*

**{**"email"**:** "keith@stones.com"**,** "name"**:** "Keith"**},**

**{**"email"**:** "charlie@stones.com"**},** *# missing "name"*

**]**

**try:**

BandMemberSchema**(**many=**True)**.load**(**user\_data**)**

**except** ValidationError **as** err**:**

pprint**(**err.messages**)**

*# {1: {'email': ['Not a valid email address.']},*

*# 3: {'name': ['Missing data for required field.']}}*

## Required Fields

Make a field required by passing required=True. An error will be raised if the the value is missing from the input to **[Schema.load()](https://marshmallow.readthedocs.io/en/stable/api_reference.html" \l "marshmallow.Schema.load" \o "marshmallow.Schema.load)**.

To customize the error message for required fields, pass a **[dict](https://python.readthedocs.io/en/latest/library/stdtypes.html" \l "dict" \o "(in Python v3.7))** with a required key as the error\_messages argument for the field.

**from** pprint **import** pprint

**from** marshmallow **import** Schema**,** fields**,** ValidationError

**class** UserSchema**(**Schema**):**

name = fields.String**(**required=**True)**

age = fields.Integer**(**required=**True,** error\_messages=**{**"required"**:** "Age is required."**})**

city = fields.String**(**

required=**True,**

error\_messages=**{**"required"**:** **{**"message"**:** "City required"**,** "code"**:** 400**}},**

**)**

email = fields.Email**()**

**try:**

result = UserSchema**()**.load**({**"email"**:** "foo@bar.com"**})**

**except** ValidationError **as** err**:**

pprint**(**err.messages**)**

*# {'age': ['Age is required.'],*

*# 'city': {'code': 400, 'message': 'City required'},*

*# 'name': ['Missing data for required field.']}*

## Partial Loading

When using the same schema in multiple places, you may only want to skip required validation by passing partial.

**class** UserSchema**(**Schema**):**

name = fields.String**(**required=**True)**

age = fields.Integer**(**required=**True)**

result = UserSchema**()**.load**({**"age"**:** 42**},** partial=**(**"name"**,))**

*# OR UserSchema(partial=('name',)).load({'age': 42})*

print**(**result**)** *# => {'age': 42}*

You can ignore missing fields entirely by setting partial=True.

**class** UserSchema**(**Schema**):**

name = fields.String**(**required=**True)**

age = fields.Integer**(**required=**True)**

result = UserSchema**()**.load**({**"age"**:** 42**},** partial=**True)**

*# OR UserSchema(partial=True).load({'age': 42})*

print**(**result**)** *# => {'age': 42}*

## Specifying Defaults

**load\_default** specifies the default deserialization value for a field. Likewise, **dump\_default** specifies the default serialization value.

**class** UserSchema**(**Schema**):**

id = fields.UUID**(**load\_default=uuid.uuid1**)**

birthdate = fields.DateTime**(**dump\_default=dt.datetime**(**2017**,** 9**,** 29**))**

UserSchema**()**.load**({})**

*# {'id': UUID('337d946c-32cd-11e8-b475-0022192ed31b')}*

UserSchema**()**.dump**({})**

*# {'birthdate': '2017-09-29T00:00:00+00:00'}*

## Handling Unknown Fields

By default, [**load**](https://marshmallow.readthedocs.io/en/stable/api_reference.html#marshmallow.Schema.load) will raise a **[ValidationError](https://marshmallow.readthedocs.io/en/stable/marshmallow.exceptions.html" \l "marshmallow.exceptions.ValidationError" \o "marshmallow.exceptions.ValidationError)** if it encounters a key with no matching Field in the schema.

This behavior can be modified with the unknown option, which accepts one of the following:

* **RAISE** (default): raise a **[ValidationError](https://marshmallow.readthedocs.io/en/stable/marshmallow.exceptions.html" \l "marshmallow.exceptions.ValidationError" \o "marshmallow.exceptions.ValidationError)** if there are any unknown fields
* **EXCLUDE**: exclude unknown fields
* **INCLUDE**: accept and include the unknown fields

You can specify unknown in the class Meta of your [**Schema**](https://marshmallow.readthedocs.io/en/stable/api_reference.html#marshmallow.Schema),

**from** marshmallow **import** Schema**,** INCLUDE

**class** UserSchema**(**Schema**):**

**class** Meta**:**

unknown = INCLUDE

at instantiation time,

schema = UserSchema**(**unknown=INCLUDE**)**

or when calling [**load**](https://marshmallow.readthedocs.io/en/stable/api_reference.html#marshmallow.Schema.load).

UserSchema**()**.load**(**data**,** unknown=INCLUDE**)**

# **Nesting Schemas**

Schemas can be nested to represent relationships between objects (e.g. foreign key relationships).

**import** datetime **as** dt

**class** User**:**

**def** \_\_init\_\_**(**self**,** name**,** email**):**

self.name = name

self.email = email

self.created\_at = dt.datetime.now**()**

self.friends = **[]**

self.employer = **None**

**class** Blog**:**

**def** \_\_init\_\_**(**self**,** title**,** author**):**

self.title = title

self.author = author *# A User object*

Use a [**Nested**](https://marshmallow.readthedocs.io/en/stable/marshmallow.fields.html#marshmallow.fields.Nested) field to represent the relationship, passing in a nested schema.

**from** marshmallow **import** Schema**,** fields

**class** UserSchema**(**Schema**):**

name = fields.String**()**

email = fields.Email**()**

created\_at = fields.DateTime**()**

**class** BlogSchema**(**Schema**):**

title = fields.String**()**

author = fields.Nested**(**UserSchema**)**

## Two-way Nesting

If you have two objects that nest each other, you can pass a callable to [**Nested**](https://marshmallow.readthedocs.io/en/stable/marshmallow.fields.html#marshmallow.fields.Nested). This allows you to resolve order-of-declaration issues, such as when one schema nests a schema that is declared below it.

For example, a representation of an Author model might include the books that have a many-to-one relationship to it. Correspondingly, a representation of a Book will include its author representation.

**class** BookSchema**(**Schema**):**

id = fields.Int**(**dump\_only=**True)**

title = fields.Str**()**

*# Make sure to use the 'only' or 'exclude'*

*# to avoid infinite recursion*

author = fields.Nested**(lambda:** AuthorSchema**(**only=**(**"id"**,** "title"**)))**

**class** AuthorSchema**(**Schema**):**

id = fields.Int**(**dump\_only=**True)**

title = fields.Str**()**

books = fields.List**(**fields.Nested**(**BookSchema**(**exclude=**(**"author"**,))))**

## Creating A Field Class

To create a custom field class, create a subclass of **[marshmallow.fields.Field](https://marshmallow.readthedocs.io/en/stable/marshmallow.fields.html" \l "marshmallow.fields.Field" \o "marshmallow.fields.Field)** and implement its [**\_serialize**](https://marshmallow.readthedocs.io/en/stable/marshmallow.fields.html#marshmallow.fields.Field._serialize) and/or [**\_deserialize**](https://marshmallow.readthedocs.io/en/stable/marshmallow.fields.html#marshmallow.fields.Field._deserialize) methods.

**from** marshmallow **import** fields**,** ValidationError

**class** PinCode**(**fields.Field**):**

*"""Field that serializes to a string of numbers and deserializes*

*to a list of numbers.*

*"""*

**def** \_serialize**(**self**,** value**,** attr**,** obj**,** \*\*kwargs**):**

**if** value **is** **None:**

**return** ""

**return** "".join**(**str**(**d**)** **for** d **in** value**)**

**def** \_deserialize**(**self**,** value**,** attr**,** data**,** \*\*kwargs**):**

**try:**

**return** **[**int**(**c**)** **for** c **in** value**]**

**except** **ValueError** **as** error**:**

**raise** ValidationError**(**"Pin codes must contain only digits."**)** **from** error

**class** UserSchema**(**Schema**):**

name = fields.String**()**

email = fields.String**()**

created\_at = fields.DateTime**()**

pin\_code = PinCode**()**