Valedictory Documentation

Release 0.8.0

Takeflight

Oct 31, 2018

Contents

Declare a schema, then validate Python dictionaries against it:

```
from valedictory import Validator, fields, InvalidDataException

class ContactValidator(Validator):
    name = fields.CharField()
    height = fields.IntegerField()
    date_of_birth = fields.DateField()

contact_validator = ContactValidator()

input_data = json.loads(request.body)

try:
    cleaned_data = contact_validator.clean(input_data)

    # Save the data
    Contact.objects.create(**cleaned_data)

except InvalidDataException as errors:
    # Handle the error
    for path, message in errors.flatten():
        print("{0}: {1}".format('.'.join(path), message))
```

CHAPTER 1

Documentation

1.1 Setup

`valedictory` is compatible with Python 3.4 or higher. It can be installed using `pip`:

```
$ pip3 install valedictory
```

1.2 Usage

All validation is done by *`Validator`* subclasses, and the fields from *`valedictory.fields`*. The schema to validate data against is defined declaratively by subclassing *`Validator`*:

```
from valedictory import Validator, fields

class ContactValidator(Validator):
    name = fields.CharField()
    height = fields.IntegerField()
    date_of_birth = fields.DateField()
```

`Validator` instances are immutable, so the same instance can be reused to validate all your data.

```
contact_validator = ContactValidator()
```

Validation is done by calling the *`clean()`* method on a *`Validator`* instance. *`clean()`* will either returned the cleaned data, or raise an *`InvalidDataException`*. *`InvalidDataException`* instances store all the validation errors found when validating the data. These can be used to create an appropriate error response.

```
from valedictory.exceptions import InvalidDataException

input_data = json.loads(request.body)

```

(continues on next page)

(continued from previous page)

```
try:
    cleaned_data = contact_validator.clean(input_data)

    # Save the data
    Contact.objects.create(**cleaned_data)

except InvalidDataException as errors:
    # Handle the error
    for path, message in errors.flatten():
        print("{0}: {1}".format('.'.join(path), message))
```

1.3 Reference

1.3.1 Validator

class valedictory.**Validator**(*fields=None*, *allow_unknown_fields=None*, *error_messages=None*, ***kwargs*)

The base class from which all custom validators are created. To define a resuable custom validator, create a new subclass of *Validator* and add any fields required:

```
from valedictory import Validator, fields

class ContactValidator(Validator):
    allow_unknown_fields = True

    default_error_messages = {
        'unknown': "I don't know what this field is"
    }

    name = fields.CharField()
    height = fields.IntegerField()
    date_of_birth = fields.DateField()

contact_validator = ContactValidator()
```

Alternatively, a validator can be defined by instantiating *Validator*:

```
contact_validator = Validator(
    allow_unknown_fields = True,
    error_messages = {
        'unknown': "I don't know what this field is"
    },
    fields={
        'name': fields.CharField(),
        'height': fields.IntegerField(),
        'date_of_birth': fields.DateField()
    },
)
```

This is useful for one-off validators, or with a *NestedValidator*.

Attributes

allow_unknown_fields = False

Whether unknown fields in the data should be treated as an error, or ignored. If

`allow_unknown_fields` is `True`, fields in the data being cleaned that do not have a corresponding field on the class cause a *`InvalidDataException`* to be thrown

The default is `False`, unknown fields are not allowed.

`fields = {name: Field()}`
A dictionary of all the fields on this validator

`default_error_messages = {'error': "Error message"}`
A dictionary of strings for each error message that can be thrown. You can override error messages by overriding this dictionary. Only the error messages being overridden need to be set, other error messages will be taken from the parent classes.

unknown Thrown when an unknown key is present in the data being validated and *`allow_unknown_fields`* is `True`.

Methods

`clean`(*data*, **args*, ***kwargs*)
Take input data, validate that it conforms to the required schema, and return the cleaned output.

If the data does not conform to the required schema, an *`InvalidDataException`* will be raised.

`error`(*code*, *params=None*, *cls=<class 'valedictory.exceptions.ValidationException'>*, ***kwargs*)
Construct a validation exception. The message will be pulled from the *`default_error_messages`* dictionary using `code` as the key. If the error message takes format parameters, pass in a dict as the `params` argument.

1.3.2 Fields

Field

`class` `valedictory.fields.`**`Field`**(*required=None*, *error_messages=None*, ***kwargs*)
The base class for all fields. By itself, *`Field`* only enforces the *`Field.required`* behaviour. Subclasses of *`Field`* add more validation rules to add more useful behaviours.

Attributes

`required = True`
Is this field required to be present in the data.

If a field is not required, and is not present in the data, it will not be present in the cleaned data either. If a field is required, and is not present in the data, a *`ValidationException`* will be thrown.

`default`
The default for this field if no value is supplied. Can be `None`. If not set, there is no default and the field will not be present in the cleaned data.

`default_error_messages`
A dictionary of messages for each error this field can raise. The default error messages can be overridden by passing an `error_messages` dict to the constructor.

required Raised when the field is not in the input data, but the field is required.

Methods

`clean`(*data*)
Clean and validate the given data.

If there is no data for the field, pass in the *`NoData`* class to signal this. If the field is required, a *`ValidationException`* will be raised. If the field is not required, *`NoData`* is returned.

error(*code*, *params=None*, *cls=<class 'valedictory.exceptions.ValidationException'>*, ***kwargs*)
Construct a validation exception. The message will be pulled from the *default_error_messages* dictionary using `code` as the key. If the error message takes format parameters, pass in a dict as the `params` argument.

TypedField

class valedictory.fields.**TypedField**(***, *required_types=None*, *excluded_types=None*, *type_name=None*, ***kwargs*)
A *Field* that requires a specific type of input, such as strings, integers, or booleans.

required_types
A tuple of acceptable classes for the data. For example, `(int, float)` would accept any number type.

excluded_types
A tuple of unacceptable classes for the data. For example, `bool`s are subclasses of `int`s, but should not be accepted as valid data when a number is expected.

type_name
The friendly name of the type for error messages.

default_error_messages

invalid_type Raised when the incoming data is not an instance of *required_types*, or is a subclass of *excluded_types*.

StringField

class valedictory.fields.**StringField**(*min_length=None*, *max_length=None*, ***kwargs*)
Accepts a string, and only strings.

min_length = 0
The minimum acceptable length of the string. Defaults to no minimum length.

max_length = inf
The maximum acceptable length of the string. Defaults to no maximum length.

default_error_messages

non_empty Raised when the input is an empty string, but *min_length* is 1.

min_length Raised when the input is shorter than *min_length*.

max_length Raised when the input is longer than *max_length*.

BooleanField

class valedictory.fields.**BooleanField**(***, *required_types=None*, *excluded_types=None*, *type_name=None*, ***kwargs*)
A field that only accepts True and False values.

NumberField

class valedictory.fields.**NumberField**(*min=None*, *max=None*, ***kwargs*)
A field that only accepts numbers, either floats or integers.

min = None
The minimum allowable value. Values lower than this will raise an exception. Defaults to no minimum value.

max = None
The maximum allowable value. Values higher than this will raise an exception. Defaults to no maximum value.

default_error_messages

min_value Raised when the value is lower than *min*.

max_value Raised when the value is higher than *max*.

IntegerField

class valedictory.fields.**IntegerField**(*min=None*, *max=None*, ***kwargs*)
A *NumberField* that only accepts integers.

FloatField

class valedictory.fields.**FloatField**(*min=None*, *max=None*, ***kwargs*)
A *NumberField* that only accepts floating point numbers.

EmailField

class valedictory.fields.**EmailField**(*min_length=None*, *max_length=None*, ***kwargs*)
A *StringField* that only accepts email address strings. The email matching regular expression only checks for basic conformance: the string must have at least one character, then an '@' symbol, then more characters with at least one dot.

default_error_messages

invalid_email Raised when the data is not a valid email address.

DateTimeField

class valedictory.fields.**DateTimeField**(*, *timezone_required=None*, ***kwargs*)
A field that only accepts ISO 8601 date time strings.

After cleaning, a `datetime.datetime` instance is returned.

timezone_required
If a timezone is required. If this is `False`, naive datetimes will be allowed.

default_error_messages

invalid_format Raised when the input is not a valid ISO8601-formatted date time

no_timezone Raised when the input does not have a timezone specified, but *timezone_required* is `True`

DateField

class valedictory.fields.**DateField**(*min_length=None*, *max_length=None*, ***kwargs*)
A field that only accepts ISO 8601 date strings.

After cleaning, a `datetime.date` instance is returned.

default_error_messages

invalid_format Raised when the input is not a valid date

TimeField

class valedictory.fields.**TimeField**(*, *timezone_required=None*, ***kwargs*)
A field that only accepts ISO 8601 time strings.

After cleaning, a `datetime.time` instance is returned.

timezone_required
If a timezone is required. If this is `False`, naive times will be allowed.

default_error_messages

invalid_format Raised when the input is not a valid ISO8601-formatted date time

no_timezone Raised when the input does not have a timezone specified, but *`timezone_required`* is `True`

YearMonthField

class valedictory.fields.**YearMonthField**(*min_length=None*, *max_length=None*, ***kwargs*)
A field that only accepts `YYYY-MM` date strings.

After cleaning, a tuple of `(year, month)` integers are returned.

default_error_messages

invalid_format Raised when the input is not a valid year-month tuple

ChoiceField

class valedictory.fields.**ChoiceField**(*choices=None*, ***kwargs*)
A field that only accepts values from a predefined set of choices. The values can be of any hashable type.

choices = set()
The field will only accept data if the value is in this set.

default_error_messages

invalid_choice Raised when the value is not one of the valid choices

ChoiceMapField

class valedictory.fields.**ChoiceMapField**(*choices=None*, ***kwargs*)
A field that only accepts values from a predefined dictionary of choices. The dictionary maps from valid input choices to the cleaned value returned.

For example:

```
>>> field = ChoiceMapField({1: 'one', 2: 'two', 3: 'three'})
>>> field.clean(1)
'one'
>>> field.clean("one")
valedictory.exceptions.ValidationException: Not a valid choice
```

would only accept one of the numbers 1, 2 or 3 as input, and would return one of the strings "one", "two", or "three".

choices = set()
The field will only accept data if the value is in this set.

default_error_messages

invalid_choice Raised when the value is not one of the valid choices

PunctuatedCharacterField

class valedictory.fields.**PunctuatedCharacterField**(*alphabet=None*, *punctuation=None*, *min_length=None*, *max_length=None*, ***kwargs*)

A field that accepts characters only from an alphabet of allowed characters. A set of allowed punctuation characters are allowed and discarded when cleaned.

alphabet
A string of all the allowed characters, not including *punctuation* characters. The cleaned output will consist only of characters from this string.

punctuation
A string of all the punctuation characters allowed. Punctuation characters will be removed from the cleaned output.

min_length = 0
The minimum length of the cleaned output data, not including punctuation characters. There is no minimum length by default.

max_length = inf
The maximum length of the cleaned output data, not including punctuation characters. There is no maximum length by default.

default_error_messages

allowed_characters Raised when characters not in *alphabet* or *punctuation* are in the input.

min_length Raised when the cleaned string is shorter than *min_length*.

max_length Raised when the cleaned string is longer than *max_length*.

RestrictedCharacterField

class valedictory.fields.**RestrictedCharacterField**(*alphabet=None*, *punctuation=None*, *min_length=None*, *max_length=None*, ***kwargs*)

A field that only allows a defined alphabet of characters to be used.

This is just a *PunctuatedCharacterField*, with *punctuation* set to the empty string.

alphabet
A string of the characters allowed in the input. If the input contains a character not in this string, a *ValidationException* is raised.

DigitField

class valedictory.fields.**DigitField**(*alphabet=None*, *punctuation=None*, *min_length=None*, *max_length=None*, ***kwargs*)
A field that only allows strings made up of digits. It is not treated as a number, and leading zeros are preserved.

CreditCardField

class valedictory.fields.**CreditCardField**(*alphabet=None*, *punctuation=None*, *min_length=None*, *max_length=None*, ***kwargs*)
Accepts credit card numbers. The credit card numbers are checked using the Luhn checksum.

The credit card number can optionally be punctuated by " -" characters.

default_error_messages

luhn_checksum Raised when the credit card is not valid, according to the Luhn checksum

ListField

class valedictory.fields.**ListField**(*field=None*, ***kwargs*)
A list field validates all elements of a list against a field. For example, to accept a list of integers, you could declare a *ListField* like:

```
class MyValidator(Validator):
    numbers = ListField(IntegerField())
```

field
The field to validate all elements of the input data against.

NestedValidator

class valedictory.fields.**NestedValidator**(*validator=None*, ***kwargs*)
Nested validators allow nesting dicts inside one another. A validator is used to validate and clean the nested dict. To validate a person with structured address data, you could make a *Validator* like:

```
class AddressValidator(Validator):
    street = StringField(min_length=1)
    suburb = StringField(min_length=1)
    postcode = DigitField(min_length=4, max_length=4)
    state = ChoiceField('ACT NSW NT QLD SA TAS VIC WA'.split())

class Person(Validator):
    name = StringField()
    address = NestedValidator(AddressValidator())
```

This would accept data like:

```
{
    "name": "Alex Smith",
    "address": {
        "street": "123 Example Street",
        "suburb": "Example Burb",
        "postcode": "7123",
        "state": "TAS"
    }
}
```

1.3.3 Exceptions

exception valedictory.exceptions.**BaseValidationException**
All validation exceptions will be subclasses of this exception.

exception valedictory.exceptions.**InvalidDataException**(*errors={}*)
Lists of validation errors for each field in a validator.

Only filled out for a field if the field has an error.

invalid_fields
A dict with the validation exceptions for all fields that failed validation. Normal field errors will be a *ValidationException* instance, while errors for *ListField* and *NestedValidator* may also be *InvalidDataException* instances.

flatten()
Yield a pair of `(path, errors)` for each error.

```
>>> list(errors.flatten())
[
    (['bar'], ['Unknown field']),
    (['foo'], ['This field can not be empty']),
]
```

If an error is a *InvalidDataException*, then errors are recursively extracted. `path` will be an array of the path to the error. `errors` will be an array of the error messages from these errors.

If validator was constructed for a shopping cart, which had user details and a list of items in a shopping cart, made using a *NestedValidator* inside a *ListField*, some possible flattened errors might be:

```
>>> list(errors.flatten())
[
    (['name'], ['This field can not be empty']),
    (['items', 2, 'quantity'], ['This must be equal to or greater than the␣
↪minimum of 1']),
]
```

exception valedictory.exceptions.**ValidationException**(*message*, *code*, ***kwargs*)
A field has failed validation.

msg
The validation error as a human readable string. This error is translatable via gettext, and should not be used for checking the type of error

code
The validation error code as a string. This can be used to check the type of error

exception valedictory.exceptions.**NoData**
Used to indicate that this field had no data supplied. This is different from having empty data, which is represented by `None`. This bypasses the bit where data is set in the output dict, so the output dict will **not** have the associated key.

1.3.4 Extensions

These extensions integrate with other Python packages to enhance valedictory.

Django fields

URLField

class valedictory.ext.django.**URLField**(*min_length=None*, *max_length=None*, ***kwargs*)
Accepts a URL as a string.

ForeignKeyField

class valedictory.ext.django.**ForeignKeyField**(*queryset*, *field='pk'*, *key_type=<class 'int'>*, ***kwargs*)
Accepts foreign keys to a Django model, and returns the model instance when cleaned.

default_error_messages

UploadedFileField

class valedictory.ext.django.**UploadedFileField**(***, *required_types=None*, *excluded_types=None*, *type_name=None*, ***kwargs*)
Accepts uploaded files

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

v

Index